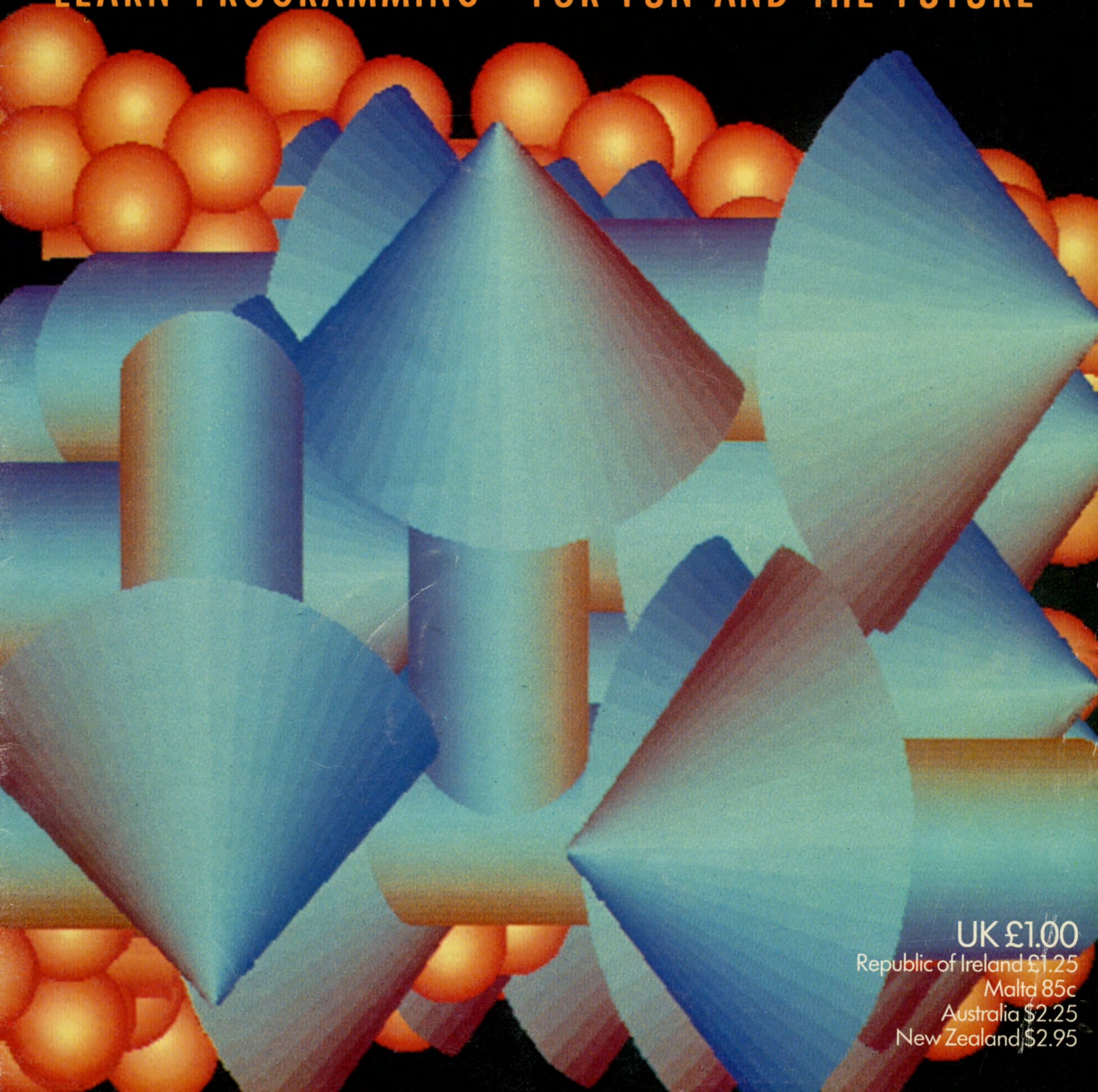
A MARSHALL CAVENDISH **14** COMPUTER COURSE IN WEEKLY PARTS

# INPUT

LEARN PROGRAMMING - FOR FUN AND THE FUTURE



UK £1.00
Republic of Ireland £1.25
Malta 85c
Australia $2.25
New Zealand $2.95

# INPUT

**Vol. 2** **No 14**



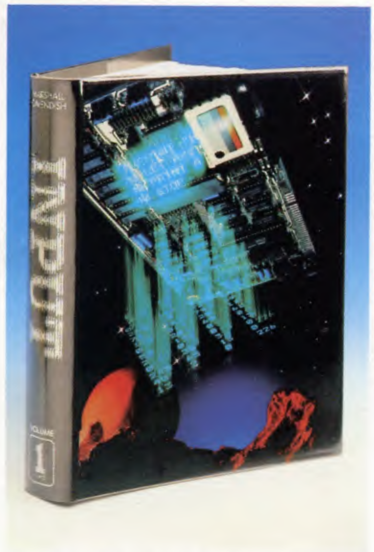
## INDEX

The last part of INPUT, Part 52, will contain a complete, cross-referenced index. For easy access to your growing collection, a cumulative index to the contents of each issue is contained on the inside back cover.

**PICTURE CREDITS**
Front cover, Digital Arts. Page 413, Howard Kingsnorth. Page 414, Tudor Art Studios. Page 416, Kuo Kang Chen. Page 421, Digital Arts. Pages 422, 424, 426, 427, Paddy Mounter. Page 428, Jeremy Gower, Chris Lyon. Page 430, Chris Lyon. Page 432, Richard Prideaux. Pages 433, 435, 438, Colin Mier. Pages 436, 437, Peter Reilly. Pages 440, 442, Artist Partners/Gino D'Achille.



Published by Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA, England. Printed by Artisan Presss, Leicester and Howard Hunt Litho, London.



*There are four binders each holding 13 issues.*

## HOW TO ORDER YOUR BINDERS

**UK and Republic of Ireland:** Send £4.95 (inc p & p) (IR£5.95) for each binder to the address below:
Marshall Cavendish Services Ltd, Department 980, Newtown Road, Hove, Sussex BN3 7DN
**Australia:** See inserts for details, or write to INPUT, Times Consultants, PO Box 213, Alexandria, NSW 2015
**New Zealand:** See inserts for details, or write to INPUT, Gordon and Gotch (NZ) Ltd, PO Box 1595, Wellington
**Malta:** Binders are available from local newsagents.

## BACK NUMBERS

Back numbers are supplied at the regular cover price (subject to availability).
**UK and Republic of Ireland:**
INPUT, Dept AN, Marshall Cavendish Services, Newtown Road, Hove BN3 7DN
**Australia, New Zealand and Malta:**
Back numbers are available through your local newsagent.

## COPIES BY POST

Our Subscription Department can supply copies to any UK address regularly at £1.00 each. For example the cost of 26 issues is £26.00; for any other quantity simply multiply the number of issues required by £1.00. Send your order, with payment to:
Subscription Department, Marshall Cavendish Services Ltd, Newtown Road, Hove, Sussex BN3 7DN
Please state the title of the publication and the part from which you wish to start.

**HOW TO PAY: Readers in UK and Republic of Ireland:** All cheques or postal orders for binders, back numbers and copies by post should be made payable to:
*Marshall Cavendish Partworks Ltd.*

**QUERIES:** When writing in, please give the make and model of your computer, as well as the Part No., page and line where the program is rejected or where it does not work. We can only answer specific queries – and please do not telephone. Send your queries to INPUT Queries, Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA.

## INPUT IS SPECIALLY DESIGNED FOR:

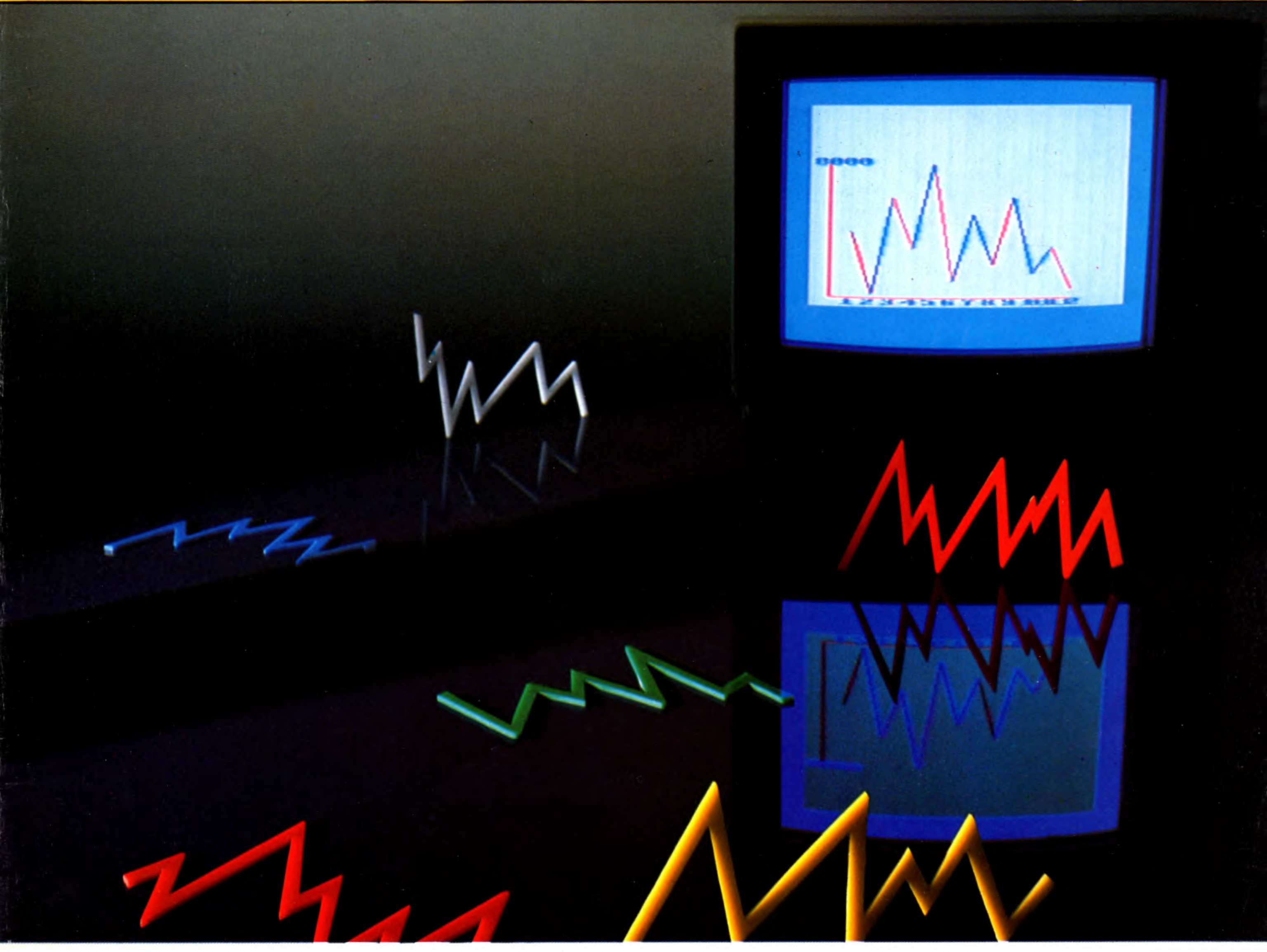
The SINCLAIR ZX SPECTRUM (16K, 48K, 128 and +), COMMODORE 64 and 128, ACORN ELECTRON, BBC B and B+, and the DRAGON 32 and 64. .

In addition, many of the programs and explanations are also suitable for the SINCLAIR ZX81, COMMODORE VIC 20, and TANDY COLOUR COMPUTER in 32K with extended BASIC. Programs and text which are specifically for particular machines are indicated by the following symbols:

SPECTRUM 16K, 48K, 128, and +
COMMODORE 64 and 128
ACORN ELECTRON, BBC B and B+
DRAGON 32 and 64
ZX81
VIC 20
TANDY TRS80 COLOUR COMPUTER

# HOW TO PLOT GRAPHS

- DRAWING GRAPHS OF MATHEMATICAL FUNCTIONS
- PLOTTING STEP GRAPHS
- SCALING AND DRAWING THE AXES



**Add your computer's data handling capacity to its graphics functions, and you have the basis for a whole range of programs to display any information as a graph or chart**

Data storage and processing is something at which computers have always excelled. But, unfortunately, it isn't so easy for a computer's user to assimilate a mass of information. If you are presented with a list of perhaps a hundred figures, it's very difficult to see an overall pattern.

For example, take a close look at the following list of numbers: 132.09, 146.2, 132.89, 123.92, 147.01, 153.47, 132.09, 138.79, 147.57, 153.9, 140.04, 142.76, 152.76, 132.6, 135.09, 146.98. Now answer the following questions:

a) Which is the highest number in the list?
b) What is its position in the list?
c) How many numbers are there below 135.5?

This isn't particularly easy, but imagine how difficult it would be if there were five times as many entries in the list.

The traditional answer to this sort of problem is to display the information graphically. When the same data is presented as a chart, it is easy to see which point is the highest, and how far along it comes. Or, by ruling off at a particular point, you can check all those values which are above or below a certain level.

Of course, you could also get your computer to make these checks for you and print out a result, but this will not give you any feel for the

overall trends—and more often than not will just result in another mass of incomprehensible data for you to sort out.

In Applications, on page 257, you have already seen one example of a program which allows you to enter data which is then displayed in the form of a bar chart on the screen. This article covers what's involved in writing your own programs to draw linear graphs. And in a later article, we take a closer look at the other types of display—histograms and pie charts, for example.

Note that the routines covered in this article are beyond the scope of the ZX81, so there are no programs for this machine. And because you cannot directly access the graphics functions in Commodore BASIC, the Commodore 64 programs use the Simons' BASIC cartridge while Vic 20 owners need to use the Super Expander cartridge.

## STANDARD GRAPH FORMS

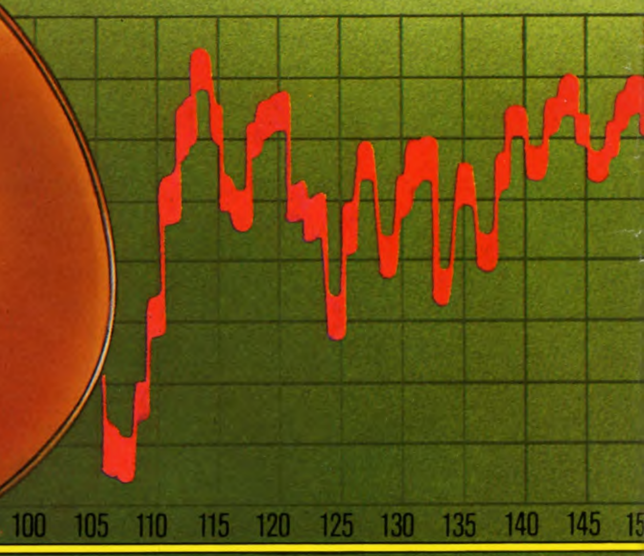
The structure of your program depends on the form of the information you wish to display. Where there is a mathematical relationship linking the data, you can make use of the mathematical functions stored in your micro's memory. For example, your micro is programmed with the trigonometric ratios—such as sine, cosine and tangent—so it is simple to plot cyclical graphs, as was explained in the article on pages 302 to 308, or as demonstrated by the following program:

```
30 PLOT Ø,76
40 FOR t=Ø TO PI*1Ø STEP .3
50 DRAW 2,COS t*15
60 NEXT t
```

```
2Ø HIRES Ø,1:MULTI 2,4,6
3Ø XX=Ø:YY=1ØØ
4Ø FOR T=Ø TO π*1Ø STEP .3
5Ø LINE XX,YY,T*5,−SIN(T)*5Ø
    +1ØØ,1
55 XX=T*5:YY=−SIN(T)*5Ø+1ØØ
6Ø NEXT T
999 GOTO 999
```

```
2Ø GRAPHIC 2
3Ø POINT 2,Ø,512
4Ø FOR T=Ø TO π*1Ø STEP .3
5Ø DRAW 2 TO T*4Ø,−SIN(T)*25Ø
    +512
6Ø NEXT T
```

```
2Ø MODE1
3Ø MOVEØ,512
4Ø FOR T=Ø TO PI*1Ø STEP.3
5Ø DRAWT*4Ø,SIN(T)*15Ø+512
6Ø NEXT
```



```
15 PMODE3,1
2Ø PCLS
25 SCREEN1,1
3Ø DRAW"BMØ,95"
4Ø FOR T=Ø TO 4Ø*ATN(1)
   STEP .Ø6
5Ø LINE-(8*T,95-4Ø*SIN(T)),
   PSET
6Ø NEXT
15Ø GOTO 15Ø
```

This program plots a sine wave, which has many applications in science and technology, in music and in graphics displays. Line 4Ø specifies a wave of five cycles (each cycle is twice PI) and Line 5Ø specifies how the wave is drawn. See pages 302 to 308 for a fuller explanation of what is going on here. By varying the values in these lines, you can alter the shape of the wave considerably.

Only a few functions are stored in your micro's memory, but you can plot any function, provided you define it in the program.

Add these next few lines to the last program and RUN it again:

```
7Ø PLOT Ø,6Ø
8Ø FOR n=1 TO 22Ø STEP 5
9Ø DRAW 5,((11Ø-n)*15)/2ØØ
1ØØ NEXT n
11Ø PLOT Ø,1ØØ
12Ø FOR n=1 TO 22Ø STEP 5
13Ø DRAW 5,-((11Ø-n)*15)/2ØØ
14Ø NEXT n
```

```
1Ø D=1:E=1ØØ
7Ø C=2
8Ø XX=8Ø-2Ø*3
85 T=-2Ø:YY=(T*T/5)*D+E
9Ø FOR T=-2Ø TO 2Ø
1ØØ LINE XX,YY,8Ø+T*3,(T*T/5)
    *D+E,C
1Ø5 XX=8Ø+T*3:YY=(T*T/5)*D+E
11Ø NEXT T
12Ø IF D=-1 THEN 999
13Ø D=-1:E=12Ø:C=3:GOTO8Ø
14Ø GOTO 8Ø
```

```
1Ø D=1:E=256
7Ø REGION 5
8Ø POINT 2,14Ø,5ØØ*D+E
9Ø FOR T=-1ØØ TO 1ØØ
1ØØ DRAW 2 TO 54Ø+T*4,T*T/2Ø
    *D+E
11Ø NEXT T
12Ø IF D=-1 THEN END
13Ø D=-1:E=E+512:REGION 2
14Ø GOTO 8Ø
```

```
1Ø D=1:E=256
7Ø GCOLØ,1
8Ø MOVE24Ø,5ØØ*D+E
9Ø FOR T=-1ØØ TO 1ØØ
1ØØ DRAW64Ø+T*4,T*T/2Ø*D+E
11Ø NEXT
12Ø IF D=-1 THEN END
13Ø D=-1:E=E+512:GCOLØ,2
14Ø GOTO 8Ø
```

```
1Ø D=1:E=64
7Ø COLOR3
8Ø DRAW"BMØ,"+STR$(INT(1Ø9*D
   +E))
9Ø FOR T=-127 TO 128
1ØØ LINE-(127+T,T*T/15Ø*D+E)
    ,PSET
11Ø NEXT
12Ø IF D=-1 THEN 15Ø
13Ø D=-1:E=E+64:COLOR2
14Ø GOTO 8Ø
```

RUN the whole program to see, as well as the sine wave, two parabolas—one inverted and the other upright. In all these cases, the data to be plotted is specified in a FOR ... NEXT loop. For example, the Spectrum program uses one loop for the sine wave and one for each of the two parabolas.

## IRREGULAR GRAPH FORMS

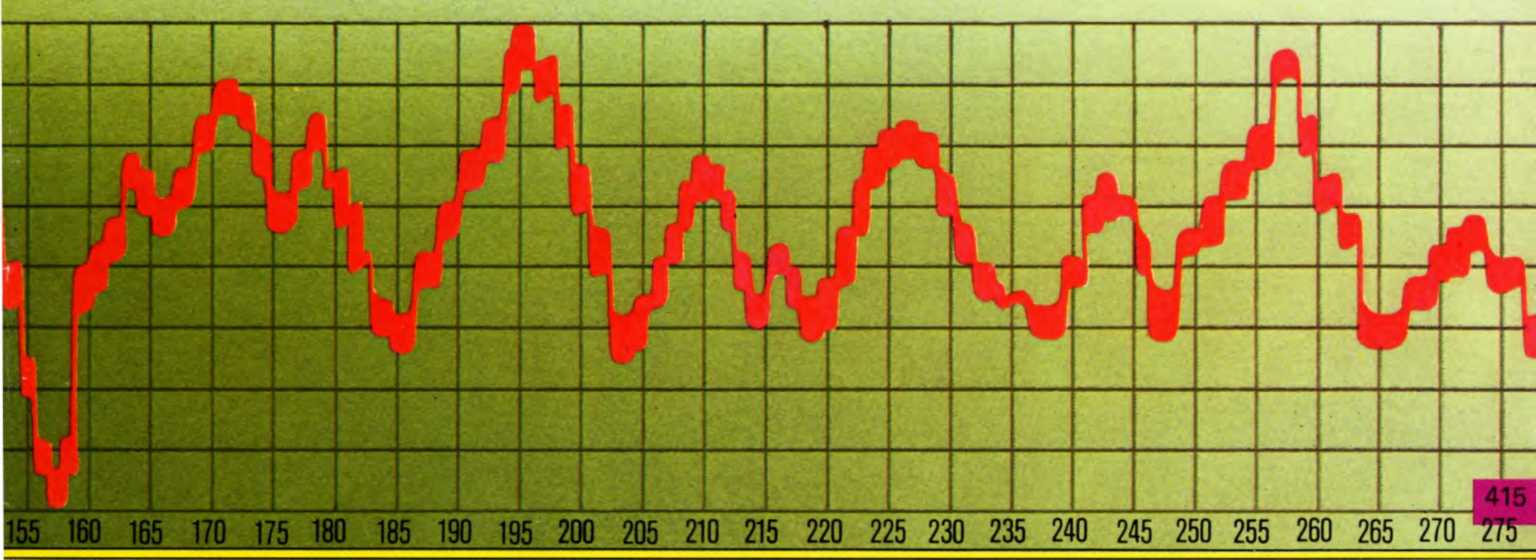
Most of the graphs you are likely to plot, however, will be from data that have no mathematical relationship. These data could be figures for annual rainfall or for the profits and sales of a business, any of which can change unpredictably or for seasonal reasons.

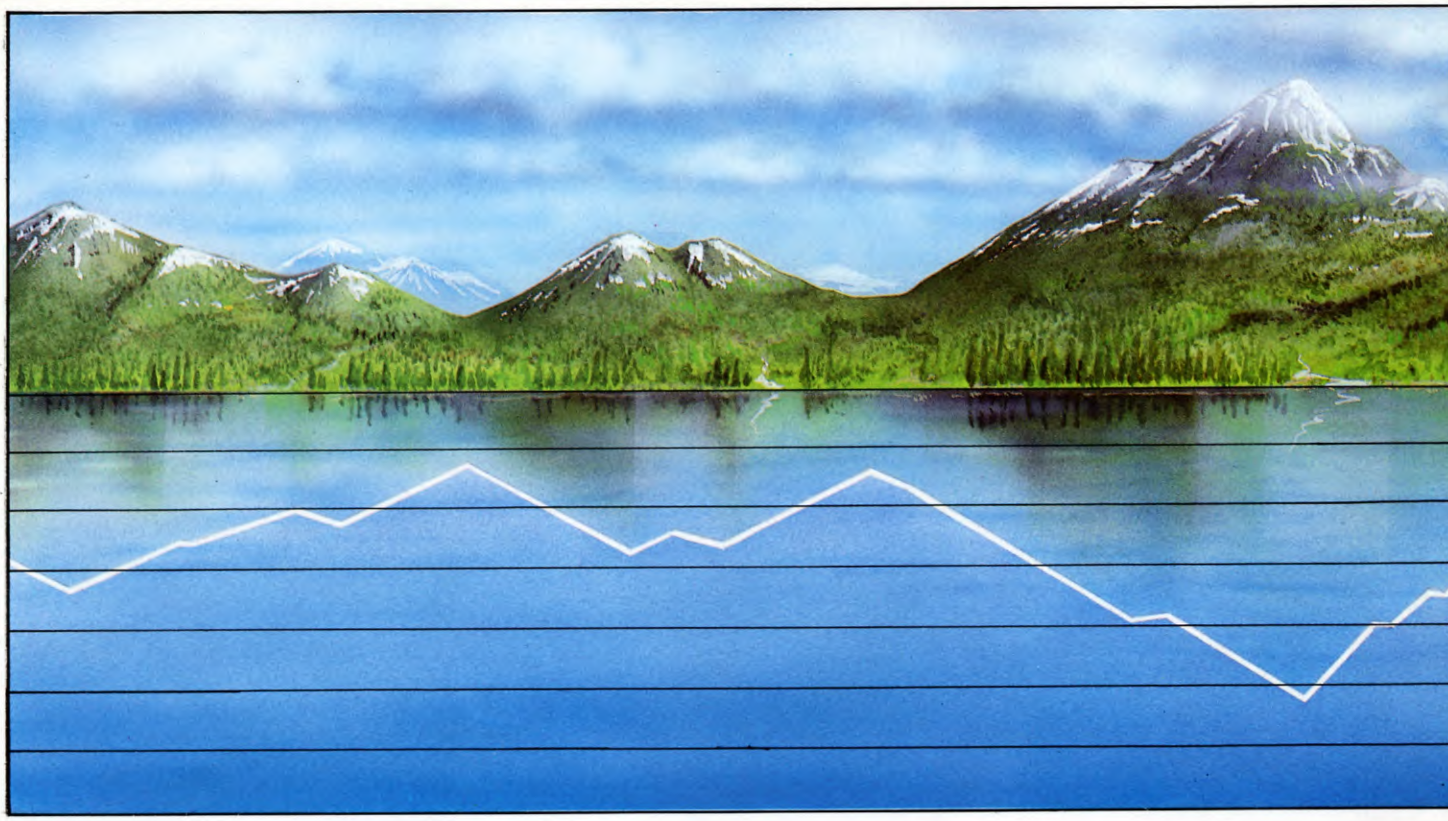
If you were plotting the graph manually, the first thing you would do is to set up the axes, so let this be the first part of the program. Type NEW then enter and RUN these lines:

```
14Ø DRAW Ø,175
15Ø PLOT Ø,Ø
16Ø DRAW 255,Ø
```

```
1Ø HIRES Ø,1:MULTI 2,4,6
14Ø LINE Ø,Ø,Ø,2ØØ,1
15Ø LINE Ø,2ØØ,36Ø,2ØØ,1
999 GOTO 999
```

```
1Ø GRAPHIC 2
14Ø DRAW 2,Ø,Ø TO Ø,1Ø23 TO 1Ø23,1Ø23
```





```
1Ø MODE 1
14Ø DRAW Ø,1Ø24
15Ø MOVE Ø,Ø
16Ø DRAW 128Ø,Ø
```

```
5 PMODE3,1
1Ø PCLS
15 SCREEN1,1
14Ø LINE(Ø,Ø) – (Ø,191),PSET
16Ø LINE (Ø,191) – (255,191),PSET
2ØØ GOTO 2ØØ
```

In these lines the program selects a graphics mode (except for the Spectrum where this is not necessary) and then places the Y-axis along the left-hand edge and the X-axis along the bottom.

This is one of the commonest arrangements for the axes, but you may have to be prepared to shift them according to the range of the data. For example, if you are plotting things like profits and loss, or temperatures, you may have negative values, so the X-axis needs to be some way up the screen, instead of at the bottom.

Another reason for shifting the axes is to leave room for numbers, division marks and labels (called legends) which are useful details to have on a graph. There is no rule about the amount of space you should leave; it depends on how much information you wish to display. Change the program as below to see the effect.

```
13Ø PLOT 2Ø,1Ø
14Ø DRAW Ø,155
15Ø PLOT 2Ø,1Ø
16Ø DRAW 235,Ø
```

```
1Ø HIRES Ø,1: MULTI 2,4,6
14Ø LINE 1Ø,Ø,1Ø,19Ø,1
15Ø LINE 1Ø,19Ø,36Ø,19Ø,1
999 GOTO 999
```

```
1Ø GRAPHIC 2
14Ø DRAW 2,1ØØ,Ø TO 1ØØ,923 TO
   1Ø23,923
```

```
1Ø MODE 1
13Ø MOVE 2ØØ,15Ø
14Ø DRAW 2ØØ,1Ø24
15Ø MOVE 2ØØ,15Ø
16Ø DRAW 128Ø,15Ø
```

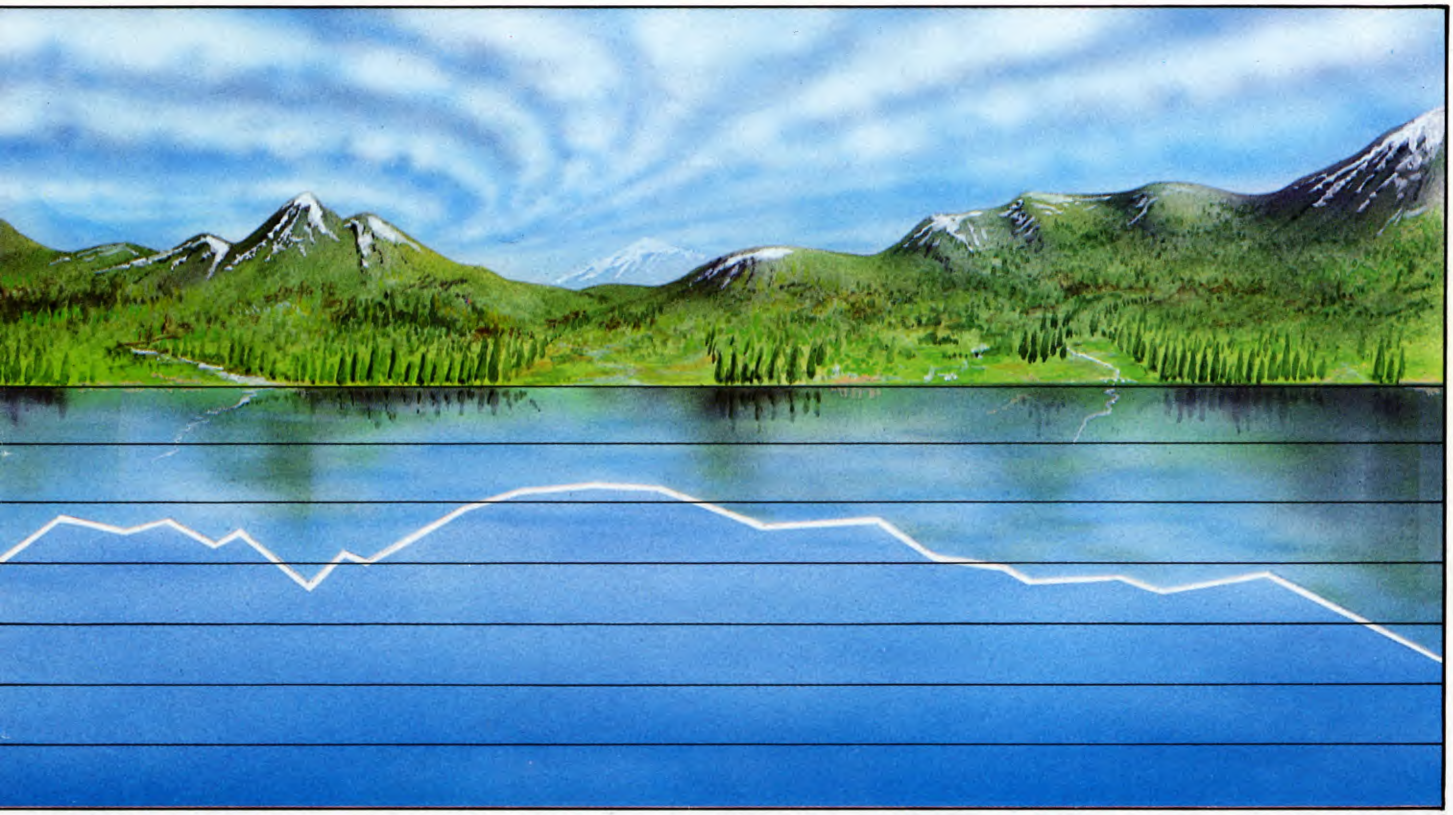
```
5 PMODE3,1
1Ø PCLS
15 SCREEN1,1
135 COLOR2
14Ø LINE(2Ø,Ø) – (2Ø,16Ø),PSET
16Ø LINE(2Ø,8Ø) – (255,8Ø),PSET
2ØØ GOTO 2ØØ
```

RUN the program and notice that there are margins to the left and bottom of the screen. To vary the size of the margin, change the values in Lines 13Ø to 16Ø, but note that on the Spectrum and Acorn micros, Lines 13Ø and 15Ø should be identical.

Enter the next few lines to continue with the program:

```
6Ø LET n=1Ø
7Ø PLOT 2Ø,n
8Ø FOR x=1 TO 12
9Ø READ y
1ØØ DRAW 18,y-n
1Ø5 LET n=y
11Ø NEXT x
1ØØØ DATA 5Ø,7Ø,6Ø,1ØØ,8Ø,12Ø,1ØØ,
   13Ø,7Ø,14Ø,9Ø,11Ø
```

```
7Ø XX=1Ø:YY=19Ø
8Ø FOR X=1Ø TO 11*1Ø+1Ø STEP 1Ø
9Ø READ Y
1ØØ LINE XX,YY,X,19Ø-Y,2
1Ø5 XX=X:YY=19Ø-Y
11Ø NEXT X
1ØØØ DATA 19Ø,1Ø,13Ø,5Ø,16Ø,Ø,1ØØ,7Ø,
   9Ø,5Ø,5Ø,Ø
```



```
7Ø POINT 2,1ØØ,923
8Ø FOR X=1ØØ TO 11*7Ø+1ØØ
   STEP 7Ø
9Ø READ Y
1ØØ DRAW 2 TO X,923-Y
11Ø NEXT X
1ØØØ DATA 5ØØ,6ØØ,41Ø,8ØØ,3ØØ,923,
   5Ø,7ØØ,5ØØ,6ØØ,25,923
```

```
7Ø MOVE 2ØØ,15Ø
8Ø FOR X=2ØØ TO 11*8Ø+2ØØ
   STEP 8Ø
9Ø READ Y
1ØØ DRAW X,Y
11Ø NEXT
1ØØØ DATA 25Ø,4ØØ,3ØØ,7ØØ,5ØØ,9ØØ,
   75Ø,95Ø,5ØØ,1ØØØ,7ØØ,85Ø
```

```
7Ø DRAW"BM2Ø,16Ø"
8Ø FOR X=2Ø TO 2Ø+11*2Ø
   STEP 2Ø
9Ø READ Y
1ØØ LINE-(X,Y),PSET
11Ø NEXT
16Ø LINE(2Ø,16Ø)-(255,16Ø),
   PSET
1ØØØ DATA 14Ø,123,148,45,1ØØ,1Ø,2Ø,
   8,45,8,45,3Ø
```

The program plots 12 points, taking the X values from Line 8Ø and Y values from Line 1ØØØ, and joins them to form a stepped graph. The first value in the DATA statement is plotted at the far left, along the Y-axis, but you could have this plotted at the first X division (month 1, maybe), by changing the initial position to which the cursor is moved. On all except the Spectrum, this position is specified at Line 7Ø, the Spectrum is in Line 6Ø.

The use of a DATA statement to store the values is a good idea when the same values are to be plotted many times, but if you wish to plot different values each time, there is a better method, using INPUTs. Add this line to the existing program. It replaces the existing line 9Ø. This works for all users of the micros covered here, except the Commodores which do not print an INPUT statement while in graphics mode.

```
9Ø INPUT "Y Value□",y: LET y=y+1Ø
```

Since Line 1Ø selects a graphics mode, an INPUT statement would not appear on the screen, so because of this, the DATA statement is the easiest method.

```
9Ø INPUT "Y VALUE□", Y:Y=Y+15Ø
```

First delete Line 15 then alter:

```
9Ø INPUT"Y VALUE□";Y
13Ø SCREEN1,1
```

When you RUN the program, it waits for you to enter the first Y value. Enter it and notice that the program executes the first round of the FOR . . . NEXT loop at Line 8Ø. Again the micro waits for you to enter the next value, and so on until all 12 values are plotted (use the ones in the DATA statement at Line 1ØØØ, now deleted).

The program works well, except that the graph is cluttered with the input routine printed by Line 9Ø (Commodore users will not have this problem). Enter the lines below to overwrite Lines 7Ø to 11Ø, then RUN the program again;

```
2Ø DIM y(12)
4Ø FOR n=1 TO 12
5Ø INPUT "Enter Y-Value□",y(n):LET
   y(n)=y(n)+1Ø
6Ø NEXT n: CLS
7Ø LET n=1Ø
8Ø PLOT 2Ø,n
9Ø FOR x=1 TO 12
1ØØ DRAW 18,y(x)-n
1Ø5 LET n=y(x)
11Ø NEXT x
```

```
10 DIM Y(11)
40 FOR N=Ø TO 11
50 PRINT "♥"N;:INPUT y(N):
   IF Y(N)<Ø OR Y(N)>19Ø
   THEN 5Ø
6Ø NEXT N:HIRES Ø,1:MULTI 2,4,6
7Ø XX=1Ø:YY=19Ø
8Ø N=Ø
9Ø FOR X=1Ø TO 11*1Ø+1Ø
   STEP 1Ø
1ØØ LINE XX,YY,X,19Ø-Y(N),RND
   (1)*3+1
1Ø5 XX=X:YY=19Ø-Y(N)
11Ø N=N+1
12Ø NEXT X
999 GOTO 999
```

```
1Ø DIM Y(11)
4Ø FOR N=Ø TO 11
5Ø PRINT "♥"N;:INPUT Y(N):
   IF Y(N)<Ø OR Y (N)>923
   THEN 5Ø
6Ø NEXT N:GRAPHIC 2
7Ø POINT 2,1ØØ,923
8Ø N=Ø
9Ø FOR X=17Ø TO 12*7Ø+1ØØ
   STEP 7Ø
1ØØ DRAW 2 TO X,923-Y(N)
11Ø N=N+1
12Ø NEXT X
```

## Microtip

### Comparing DATA?

The ability to scale a graph can be particularly useful when you wish to display more than one set of data on the screen at one time. For example, you can compare data for both halves of a year by drawing a graph for the first six months at the top of the screen, and one for the other months at the bottom. You could read in the 12 items of data into a single array, as in the program, then step through the first six and carry out the drawing routine, then repeat for the next six. The only other considerations are to specify the starting position for each graph, and to modify the axes-drawing routines, so that you draw two instead of one axis. Alternatively, you could have a single X-axis across the centre of the screen against which both of the graphs can be read.

```
2Ø DIM Y(11)
4Ø FOR N=Ø TO 11
5Ø INPUT "ENTER Y-VALUE□"Y(N):
   Y(N)=Y(N)+15Ø
6Ø NEXT : CLS
7Ø MOVE 2ØØ,Y(Ø)
8Ø N=1
9Ø FOR X=28Ø TO 11*8Ø+2ØØ STEP 8Ø
1ØØ DRAW X,Y(N)
11Ø N=N+1
12Ø NEXT
```

```
2Ø DIMY(11)
4Ø FORN=ØTO11
5Ø INPUT"ENTER Y-VALUE□";
   Y(N)
6Ø NEXT
7Ø DRAW"BM2Ø,"+STR$(INT(Y(Ø)))
8Ø N=Ø
9Ø FOR X=2Ø TO 11*2Ø+2Ø STEP 2Ø
1ØØ LINE-(X,Y(N)),PSET
11Ø N=N+1
12Ø NEXT
```

When you RUN the program, you first loop through the input routine from Line 4Ø to 6Ø 12 times entering the Y values. These values are stored in the array Y() as variables Y(1) through Y(12) on the Spectrum and Y(Ø) through Y(11) on the others. If instead of 12 values you wanted some other number, you would merely redimension the array for the required number. The screen is then cleared and the cursor is moved to the first X,Y position (Lines 7Ø and 8Ø). The rest of the program plots the graph from X values at Line 9Ø and Y values at Lines 1ØØ to 11Ø.

## SCALING FACTORS

The figures for this graph were chosen so that they fit conveniently within the range of the axes and the screen. However, the values you wish to plot will rarely be just right, but either too large or too small—so you need to scale them to bring them within range. The simplest method of scaling is to study the values and decide the factor by which they should be multiplied to bring them within range. Here is the complete new program, with all the changes made:

```
2Ø DIM y(12)
3Ø LET xs=2: LET ys=1/5Ø
4Ø FOR n=1 TO 12
5Ø READ y(n)
6Ø NEXT n: CLS
7Ø LET n=8
8Ø PLOT 4Ø,n
9Ø FOR x=1 TO 12
1ØØ DRAW 8*xs,(y(x)-n)*ys
1Ø5 LET n=y(x)
11Ø NEXT x
13Ø PLOT 4Ø,8
14Ø DRAW Ø,167
15Ø PLOT 4Ø,8
16Ø DRAW 21Ø,Ø
1ØØØ DATA 4ØØØ,2ØØØ,6ØØØ,3ØØØ,
   8ØØØ,4ØØØ,5ØØØ,2ØØØ,6ØØØ,15ØØ,
   3ØØØ,5ØØ
```

```
1Ø HIRES Ø,1:MULTI 2,4,6
2Ø DIM Y(12)
3Ø XS=1Ø:YS=1/6Ø
4Ø FOR N=1 TO 12
5Ø READ Y(N)
6Ø NEXT N
8Ø XX=1Ø+XS:YY=19Ø-(YS*Y(1))
9Ø FOR X=1 TO 12
1ØØ LINE XX,YY,1Ø+X*XS,19Ø-
   (Y(X)*YS),RND(1)*3+1
1Ø5 XX=1Ø+X*XS:YY=19Ø-
   (Y(X)*YS)
11Ø NEXT X
14Ø LINE 1Ø,Ø,1Ø,19Ø,1
15Ø LINE 1Ø,19Ø,36Ø,19Ø,1
999 GOTO 999
1ØØØ DATA 4ØØØ,2ØØØ,6ØØØ,3ØØØ,
   8ØØØ,1ØØØ,5ØØØ,2ØØØ,6ØØØ,15ØØ,
   3ØØØ,5ØØ
```

```
1Ø GRAPHIC 2
2Ø DIM Y(12)
3Ø XS=7Ø:YS=1/1Ø
4Ø FOR N=1 TO 12
5Ø READ Y(N)
6Ø NEXT N
8Ø POINT 2,1ØØ+XS,923-(YS*Y(1))
9Ø FOR X=1 TO 12
1ØØ DRAW 2 TO 1ØØ+X*XS,923-
   (Y(X)*YS)
11Ø NEXT X
14Ø DRAW 2,1ØØ,Ø TO 1ØØ,923 TO 1Ø23,923
1ØØØ DATA 4ØØØ,2ØØØ,6ØØØ,3ØØØ,
   8ØØØ,1ØØØ,5ØØØ,2ØØØ,6ØØØ,15ØØ,
   3ØØØ,5ØØ
```

```
1Ø MODE 1
2Ø DIM Y(11)
3Ø XS=5Ø:YS=.1
4Ø FOR N=Ø TO 11
5Ø READ Y(N)
6Ø NEXT
7Ø VDU 29,2ØØ;15Ø;
8Ø MOVE Ø,YS*Y(Ø)
9Ø FOR X=Ø TO 11
1ØØ DRAW X*XS,Y(X)*YS
```

```
120 NEXT
130 MOVE Ø,Ø
140 DRAW Ø,1024
150 MOVE Ø,Ø
160 DRAW 1280,Ø
1000 DATA 4000,2000,6000,3000,
   8000,4000,5000,2000,6000,1500,
   3000,500
```

```
5 PMODE3,1
10 PCLS
20 DIMY(12)
30 XS=20:YS=1/20
40 FORN=1 TO 12
50 READ Y(N)
60 NEXT
80 DRAW"BM20,"+STR$(INT(YS*Y(1)))
90 FOR X=1 TO 12
100 LINE-(X*XS,Y(X)*YS),PSET
110 NEXT
130 SCREEN1,1
135 COLOR2
140 LINE(20,Ø)-(20,160),PSET
160 LINE(20,160)-(255,160),PSET
200 GOTO 200
1000 DATA 1600,1000,2500,3000,
   3200,2000,2500,1000,3000,750,
   1500,250
```

The program plots a graph of the values in the DATA statement at Line 1ØØØ. This method of input is used to save you having to type in the values repeatedly, but you could revert to the other method as explained above.

The scale factors are set at Line 3Ø. Line 8Ø moves the cursor to the first X,Y position, and the points are plotted by the loop at Lines 9Ø to 11Ø. Lines 13Ø to 16Ø draw the axes.

This graph does not make full use of the space available on the screen, so it might be worth changing the scale factors at Line 3Ø. Try halving them or doubling them; then RUN the program to see the different effects. To scale the axes as well as the graph, add the scale factor to the lines that draw them as below:

```
140 DRAW Ø,2000*ys
160 DRAW 12*xs,Ø
```

```
140 LINE 10,190-8000*YS,10,190,1
150 LINE 10,190,12*XS+10,190,1
```

```
140 DRAW 2,100,923-8000*YS TO 100,923
   TO 12*XS+100,923
```

```
140 DRAW Ø,8000*YS
160 DRAW 11*XS,Ø
```

```
140 LINE(20,Ø)-(20,3200*YS),
   PSET
160 LINE(20,YS*3200)-(255,
   YS*3200),PSET
```

RUN the program and notice the axes extend only to the maximum values. The effect will be more noticeable if you replace the initial values in Line 3Ø. Now you can cope with any value, provided you change the scale values at Line 3Ø accordingly. Try some different values in the DATA statement and practise varying the scale factors. With very large numbers, you need to leave a larger margin along the Y-axis. Alternatively, you could retain the same margin, but PRINT a factor (for example, x1Ø) by which values along the Y-axis are multiplied.

To place some legends on the graph, enter the next few lines and RUN the program:

```
210 FOR x=Ø TO 12 STEP 2
220 PRINT AT 21,x*2+4;x
230 NEXT x
300 FOR y=Ø TO 8000 STEP 2000
310 PRINT AT (8000-y)/400,Ø;y
320 NEXT y
```

Lines 21Ø to 23Ø loop through the even numbers from Ø to 12, which are PRINTed just below the X-axis by Line 22Ø. Similarly, the numbers along the Y-axis are PRINTed by Lines 3ØØ to 32Ø.

```
106 TEXT 2+X*XS,191,STR$
   (X),3,1,3
120 TEXT 2,50,"8000",3,1,7
```

Line 1Ø6 PRINTs the numbers Ø to 12 just below the X-axis, and Line 12Ø PRINTs the largest value (8ØØØ) at the top of the Y-axis.

```
120 CHAR 18,2,"Ø"
122 CHAR 18,10,"6"
124 CHAR 18,17,"12"
126 CHAR 2,1,"8000"
```

Lines 12Ø, 122 and 124 PRINT Ø, 6 and 12 just below the X-axis, and Line 126 PRINTs the largest value at the top of the Y-axis. You'll have to change these numbers to suit each set of DATA.

```
200 VDU 5
210 FOR X=2 TO 12 STEP 2
220 MOVE (X-1)*XS-24,-15
230 PRINT;X
240 NEXT
300 FOR Y=2000 TO 8000 STEP
   2000
310 MOVE -180,Y*YS
320 PRINT;Y;"□ -"
330 NEXT
340 VDU 4
```

Lines 21Ø to 24Ø loop through the even numbers from 2 to 12, which are PRINTed just below the X-axis by Line 23Ø. The numbers are PRINTed at positions given by Line 22Ø. Notice that the negative quantities are necessary to align the numbers with the axis. Similarly, the numbers along the Y-axis, in this case 2ØØØ to 8ØØØ in steps of 2ØØØ, are PRINTed by Lines 3ØØ to 33Ø. Change these numbers for different sets of DATA.

Placing numbers on the graphics screen of these machines is a little more difficult. You cannot PRINT the standard character set, but instead have to DRAW each letter individually. The routine to do it is too long to be reproduced here, but it has been dealt with on page 192 and can be incorporated into the program with suitable adjustment to the line numbers.

### Q+A

**Could the listings be modified so that a mixture of positive and negative values can be entered and plotted on the same graph?**
The input routine can cope, as it stands, with negative and positive values, but the routines that position the axes and scale the data will need to be modified. The main difficulty is not to decide which are the maximum and minimum values, but to decide where to place the X-axis. If you are prepared to position it for each different set of data, the modification is simple. Scan the data, insert suitable scale factors and start to draw at the origin of the axes.

## FINISHING TOUCHES

All that remains is to add the finishing touches, such as colour and labels. These are not essential, but they help to make an attractive display. As an exercise, try adding these yourself, using GCOL, COLOUR and PRINT TAB or PRINT AT statements. These commands and how to use them are covered for each machine on pages 84 to 91 and 117 to 123.

# COMMODORE KEYBOARD SYMBOLS

**The uses of Commodore graphics symbols extend far beyond just making pictures. They are also frequently employed as a form of programming shorthand**

One of the most perplexing problems facing a newcomer to Commodore computers is the extensive use of on-board graphics, many of which find their way into program listings as symbols for other functions.

## ROM GRAPHICS

The Commodores have a rich selection of on-board ROM (read only memory) graphics which are accessible for any kind of screen display. What you can access depends on which screen display mode you're in: upper case (capitals) and graphics mode (referred to here as U + G), or upper case and lower case mode (U + L). A greater number are available when the computer is in U + G mode—to which it is set on power up. You can toggle out of this mode and into U + L mode by pressing the [SHIFT] key and the Commodore key—[C=] simultaneously.

In direct mode you simply press either the [SHIFT] or the [C=] key in conjunction with the required key to obtain one of the two graphics shown on the front of the key.

With [C=] you can get the graphics shown on the left hand of the key whether the display is in U + G or U + L modes. The [SHIFT] key naturally produces capital letters when used in U + L mode, but can be used to access the right hand key graphics in U + G mode.

If you are unfamiliar with the ROM graphics, switch on your computer and try out the various keys. Pressing a key on its own should result in its face 'value' being displayed—but with two important exceptions which we'll get to shortly (quote and insert modes). So avoid pressing ["] or using [INST] for the moment. Now press the same keys while holding down the [C=] key. Then do the same while holding down the [SHIFT] key. A whole series of graphics should be displayed. Don't clear the screen yet.

Now toggle the display into U + L mode and note that only the graphics which had been produced with the [C=] key remain unchanged.

Direct mode graphics serve no function other than to display what's available in ROM although this can be useful if you're using them actually within a program to construct an appealing screen display. This short program displays what's available:

[C=]

```
10 FOR Z=0 TO 255
20 POKE 1024+Z, Z
30 POKE 55296+Z, 1
40 NEXT Z
```

[C=]

```
10 FOR Z=0 TO 255
20 POKE 1024+Z, Z
30 POKE 38400+Z, 0
40 NEXT Z
```

As you can see when you RUN this, there are two sets of characters—the second series simply takes the reverse form of the first.

## PROGRAM GRAPHICS

The simplest way to use graphics from within a program is to incorporate them within PRINT statements as in the helicopter program on page 31. But this may be laborious.

Each of the graphic characters has its own particular Commodore ASCII code, and you can use this value to create screen displays using CHR$ (see pages 314 to 320). This short program prints out 63 of the graphics characters available from the keyboard:

```
10 LO=96 : HI=127
20 FOR LP=LO TO HI
30 PRINT LP, CHR$(LP)
40 GET A$: IF A$="" THEN 40
50 NEXT LP
60 IF LO=161 THEN STOP
70 LO=161 : HI=191
80 GOTO 20
```

Every time you press a key you'll see another graphics character along with the ASCII value used as the CHR$ argument.

A third way of accessing graphics from within a program is to use Commodore's special screen display codes—as used for the first set of programs in this article.

These screen codes can take the value 0 to 255, the reverse graphics used as symbols having values above 128. There are actually two character sets—these relate to the U + G and U + L modes—and only one set at a time can be displayed on the screen. Note that colour memory has to be POKEd with a suitable colour at each location where a graphic appears. Remember that the screen code values are different from the ASCII code values.

## QUOTE MODE

Low resolution graphics are useful for supplementing high resolution graphics, but are no alternative. And this is really a secondary role to the use of symbolic graphics, obtained in a line such as this by typing [SHIFT] and [CLR/HOME] after the first quotes:

```
10 PRINT "♥"
```

The reverse heart symbol makes an appearance only after you've entered quote mode or insert mode. You enter quote mode simply by typing ["] and exit by typing ["] again. The next time you type ["], you re-enter quote mode and exit it with the next—and so on. CLR the screen and try typing in the line without the first set of quotes!

While you've got a clear screen, try something else: enter insert mode by pressing [SHIFT] and [INST/DEL] keys simultaneously—then press [SHIFT] and [CLR/HOME] simultaneously. Lo and behold, the reverse heart again!

Enter quote mode—you can do this in direct mode simply by typing ["]—and find out what symbols you get when [SHIFT]ed and un[SHIFT]ed cursor keys are pressed, when the [CTRL] key is used to access the colours and reverse (obtained on keys 1 to 0) and when the function keys are pressed.

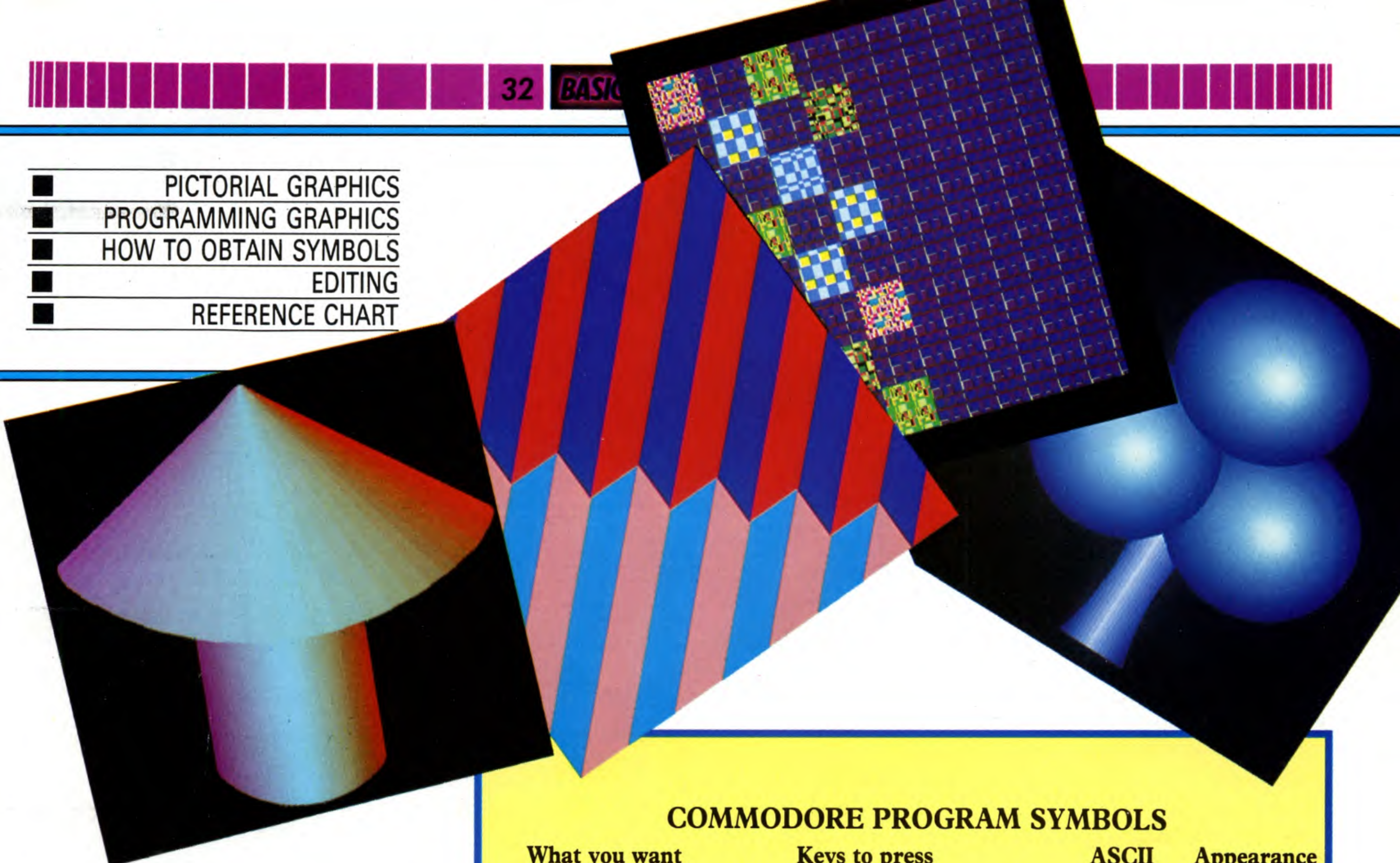
## EDITING PROGRAMS

The effect of quote mode and insert mode are, superficially, identical—but the latter works only once after going into insert mode. Quote mode works until the next set of quotes or until a program line terminates.

By itself, this is not really much of a problem. Things change though when it comes to editing a program line. For example, press the cursor down key ten times after you enter the first quotes in the following line:

```
10 PRINT "■■■■■■■■■■
   HELLO"
```

Press [RETURN] to enter the line, then return to its midpoint as if to do some editing—say, to change some of these to cursor right symbols



so that HELLO is displayed away from the left hand edge of the screen.

All that happens when you try overwriting the cursor down symbols is the cursor moves to the right. Matters don't improve if you try deleting unwanted symbols first.

There are two ways you can edit a line such as this. It's simplest actually to enter quote mode again by placing the cursor over the first quote marks. Then retype the quotes and the rest of the PRINT statement, including the amendments you wish to make.

Alternatively, position the cursor to delete unwanted symbols and when you've done this use [INST] to create extra spaces for the new symbols or characters you wish to incorporate. Each use of [INST] allows one entry in insert mode.

If, during the course of editing, you lose track of what you are doing or, for instance, you enter quote or insert mode unintentionally, simply [RETURN] the line and start editing it again. Or retype the whole line from scratch.

## TOKENS

The last thing for which the graphics symbols are used on the 64 is for abbreviating the keywords. Most of the reserved words have an abbreviation, and generally this consists of typing the first letter, and then [SHIFT] and the second letter. A full list of the abbreviations, how they look, and what to press to get them, is given on pages 130 and 131 of the user manual.

## COMMODORE PROGRAM SYMBOLS

| What you want | Keys to press | ASCII | Appearance |
|---|---|---|---|
| **Cursors:** | | | |
| cursor down | CRSR ⇓ | 17 | |
| cursor up | SHIFT and CRSR ⇑ | 145 | |
| cursor right | CRSR ⇒ | 29 | |
| cursor left | SHIFT and CRSR ⇐ | 157 | |
| **CLR/HOME:** | | | |
| Cursor to top left | CLR/HOME | 19 | |
| Ditto, plus clear | SHIFT and CLR/HOME | 147 | |
| **INST/DEL:** | | | |
| Delete character | CTRL and T | 2Ø | |
| Insert character | SHIFT and INST/DEL | 148 | |
| **Reverse mode:** | | | |
| Reverse on | CTRL and 9 | 18 | |
| Reverse off | CTRL and Ø | 146 | |
| **Text Colour:** | | | |
| Black | CTRL and 1 | 144 | |
| White | CTRL and 2 | 5 | |
| Red | CTRL and 3 | 28 | |
| Cyan | CTRL and 4 | 159 | |
| Purple | CTRL and 5 | 156 | |
| Green | CTRL and 6 | 3Ø | |
| Blue | CTRL and 7 | 31 | |
| Yellow | CTRL and 8 | 158 | |
| Orange | [C=] and 1 | 129 | |
| Brown | [C=] and 2 | 149 | |
| Light Red | [C=] and 3 | 15Ø | |
| Dark Grey | [C=] and 4 | 151 | |
| Medium Grey | [C=] and 5 | 152 | |
| Light Green | [C=] and 6 | 153 | |
| Light Blue | [C=] and 7 | 154 | |
| Light Grey | [C=] and 8 | 155 | |

N.B. The last eight items in column two will not work on the Vic 2Ø.

# ADVENTURES- THE NEXT STEP

**Adventures are like cigarettes—they're addictive, can be bought packaged, or you can roll your own. Here's how to use the *INPUT* adventure as a basis for your own**

By now you should have a fully functioning adventure game stored on tape. In exploring its development, you have seen how all the elements which make it up were brought together, starting with just a bare outline of a story. And this time you'll see how that game can be used as a basis for your own home-grown adventures.

Some hints about altering the adventure have been scattered throughout the series of articles, but this time you'll see in greater depth what has to be done.

It won't always be possible to be specific about the alterations, because many of them will depend totally on the adventure you are writing, but many alterations are simple to do if you follow the instructions later in the article. Some of the techniques may seem a little daunting at first, but if you try to write a simple, short adventure to start with you should soon pick up the principles. In the early stages don't try to make too many alterations simultaneously, just work through the sections of this article systematically and you shouldn't go far wrong.

If you are a little puzzled by some of the BASIC in the program, you can try looking in Basic Programming where many of the more common keywords have already been covered.

Vic 20 and 16K Spectrum owners will not be able to extend the adventure too much because they will soon run out of memory. But that is not to say that there is no scope at all for extending the events in the adventure, or that adventures with many more locations cannot be written. Should you wish to write an adventure with more rooms on these machines, for example, then it would be quite possible to trade off some other feature in favour of the extra rooms. It really depends on what you want from your adventure.

## YOUR OWN ADVENTURE THEMES

Before you can write your own adventures, you will have to invent a suitable plot or story.

The structure of a successful adventure story is usually a very traditional one—there is a beginning, a middle and an end, with a structure imposed by the order in which the puzzles are meant to be solved. It's lucky, though, that you don't really have to be an Agatha Christie to write adventure games. There are lots of existing sources for ideas, as you saw earlier, but to make it a little easier in the early stages, here are a few suggestions.

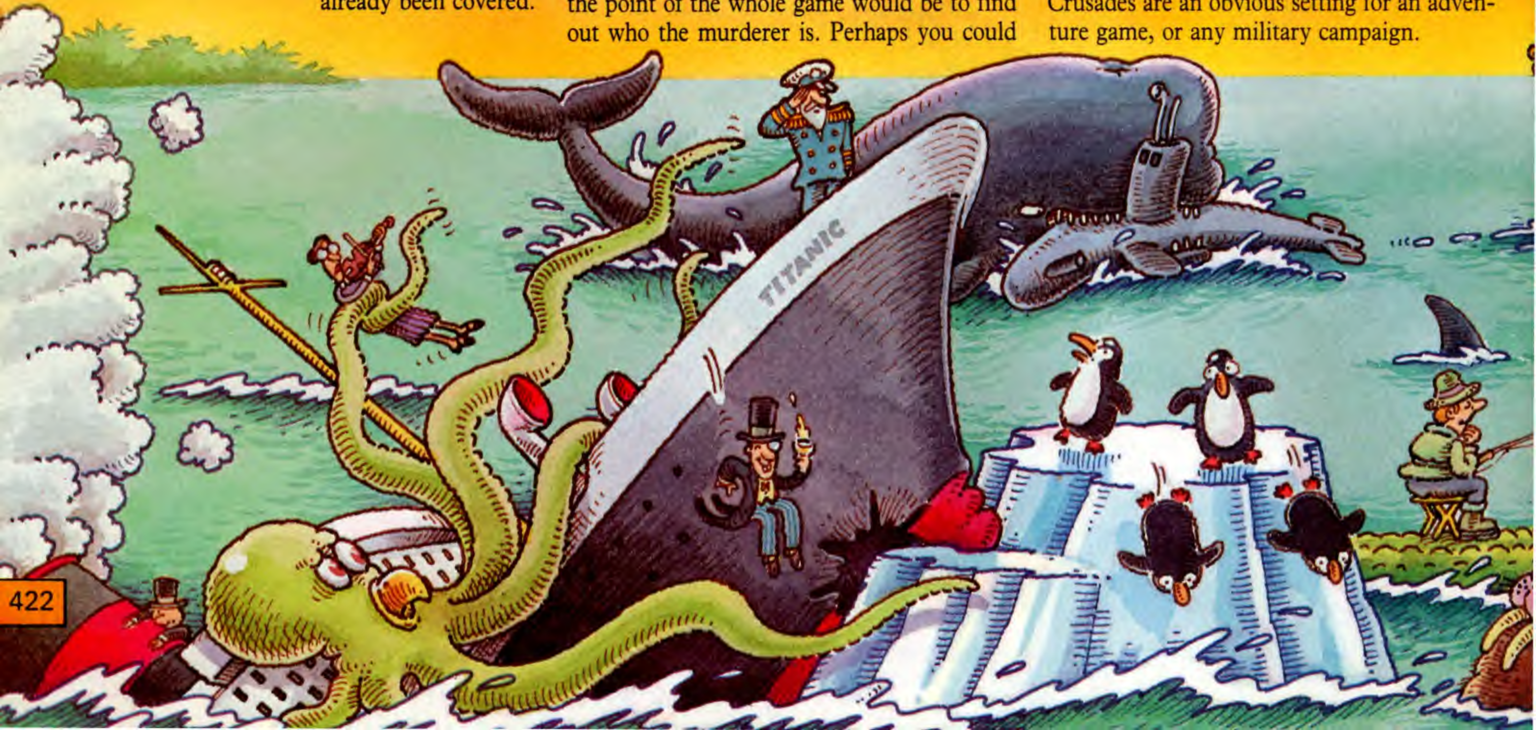
You could structure an adventure around a Whodunnit. The start could be a room with a dead body with a knife sticking out of it, and the point of the whole game would be to find out who the murderer is. Perhaps you could use a butler character instead of the tax inspector. His role could be either to help or hinder the adventurer.

There are various ways you could use a castaway theme in your adventure. Try using a traditional pirate-type castaway, or you could have your adventurer being the sole survivor of a jumbo jet crash. Setting the adventure in the future, you could have a disaster in space leading to the adventurer being marooned on a hostile planet light years from civilisation with a defective rocket. The object of the adventure could be to contrive some way to escape. The locals could very well be hostile, and there is plenty of scope for imaginative (and hidden) escape routes.

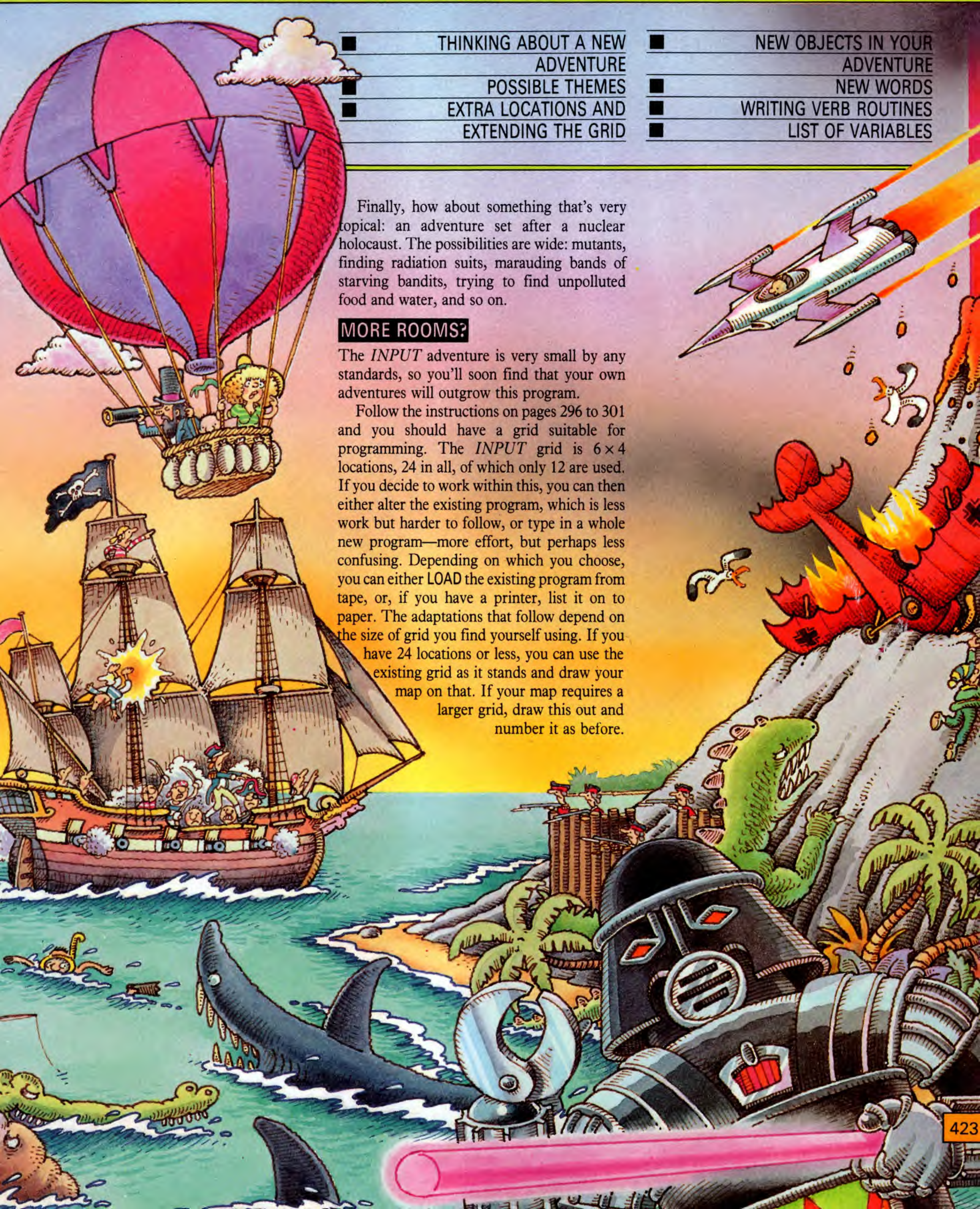
Other escape plots could include escaping from Alcatraz, or Colditz or Dartmoor—you name it. Sources of inspiration could be any one of a number of 'Escape from . . .' books that have been written—and if you can get a map of the real place, so much the better for planning out your locations.

Spies—either traditional ones, or even industrial espionage, stealing a rival's computer design, perhaps—are likely to be a very happy hunting ground for the adventure writer.

You could plunder stories from history. The Crusades are an obvious setting for an adventure game, or any military campaign.



- THINKING ABOUT A NEW ADVENTURE
- POSSIBLE THEMES
- EXTRA LOCATIONS AND EXTENDING THE GRID
- NEW OBJECTS IN YOUR ADVENTURE
- NEW WORDS
- WRITING VERB ROUTINES
- LIST OF VARIABLES



Finally, how about something that's very topical: an adventure set after a nuclear holocaust. The possibilities are wide: mutants, finding radiation suits, marauding bands of starving bandits, trying to find unpolluted food and water, and so on.

## MORE ROOMS?

The *INPUT* adventure is very small by any standards, so you'll soon find that your own adventures will outgrow this program.

Follow the instructions on pages 296 to 301 and you should have a grid suitable for programming. The *INPUT* grid is 6 × 4 locations, 24 in all, of which only 12 are used. If you decide to work within this, you can then either alter the existing program, which is less work but harder to follow, or type in a whole new program—more effort, but perhaps less confusing. Depending on which you choose, you can either LOAD the existing program from tape, or, if you have a printer, list it on to paper. The adaptations that follow depend on the size of grid you find yourself using. If you have 24 locations or less, you can use the existing grid as it stands and draw your map on that. If your map requires a larger grid, draw this out and number it as before.

Having sorted out the grid you can progress to entering your own set of location descriptions on the machine. They should replace the existing location descriptions after Line 5ØØØ.

Follow each location description with the line containing the possible exits as in the original program. The variables N, S, E, and W correspond to north, south, east, and west. They can be set to Ø or 1—Ø means that there is no exit in that direction, while 1 means that there is a way out. The extra effort of typing in REM lines with the location numbers is well worth it.

Next change the ON ... GOSUBs in Lines 33Ø to 35Ø. The first number following GOSUB in Line 33Ø is the line number where the computer will find the description of location 1. If there isn't a location 1 in the adventure—you don't have to use all the squares on the grid—then zero is entered instead. The next number is the line number of the description of the second location, and so on. There must be a number for all of the locations in the grid.

## MOVING AROUND

If you have designed an adventure which is based on a different size grid from that used for the *INPUT* adventure you'll need to alter the movement routine at Lines 1ØØØ to 1Ø4Ø. More specifically, the north and south lines—Lines 1Ø1Ø and 1Ø3Ø—will have to be altered if the grid is no longer six squares wide, since you add or subtract six to change row on the grid. Simply count how many squares there are across the top of the grid and change the figure 6 to the width of your grid.

## THE OBJECTS

The objects in your new adventure will be different from those in the *INPUT* adventure, so you'll have to make quite extensive changes to Lines 16Ø to 26Ø.

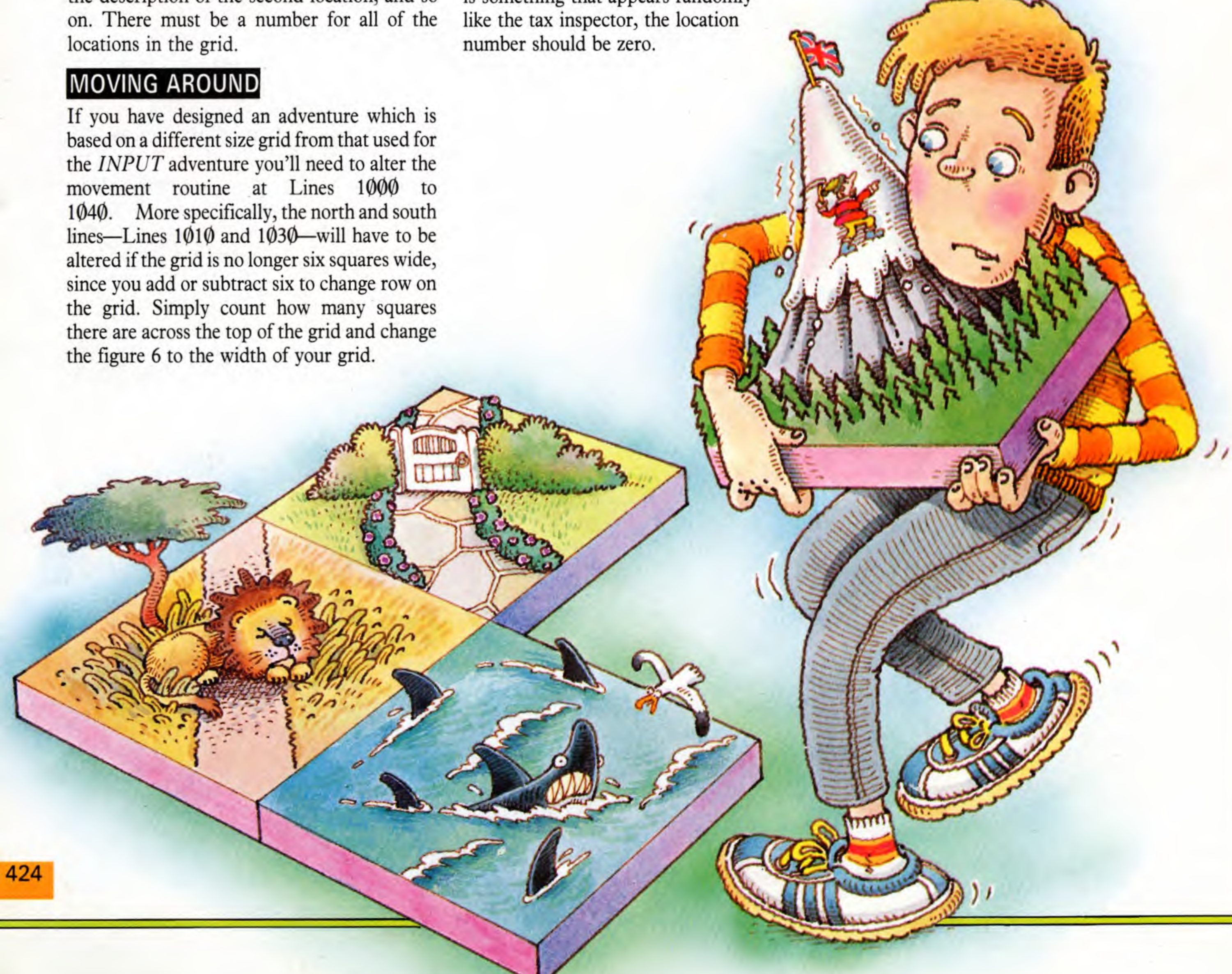
Count how many objects you are going to use in the new adventure. This number gives the value of NB and it should be the first piece of data in Line 2ØØ and will be used to DIMension arrays in Line 18Ø, and to set up FOR ... NEXT loops elsewhere in the program.

It's clearest to use a separate program line for each object, but if you've written a game needing a large number of objects, you may find that it's better to have more than one object per line. Whichever way you decide to enter your data the order must be right, as each of the three pieces of data are fed into different arrays. The order, then, is: location number, short description, long description. If the object only appears later on in the adventure, perhaps after the adventurer has found it, or it is something that appears randomly like the tax inspector, the location number should be zero.

## NEW WORDS

Make a list of all the instructions that you will be expecting the adventurer to give during the adventure. The list should include single words, such as the directions and HELP and INVENTORY, and two-word commands like GET LAMP or KILL LANDLORD.

The two-word inputs are split into V$ and N$—verb and noun, although these are not always strictly according to their grammatical definition. You are interested in all the single words and the first word in each of the pairs. For the program's purposes these are the verbs—V$s. Group the verbs together according to their meanings—CHEW and EAT, or SMELL and SNIFF should be grouped together, for example. Each of these groups will need a number, so note that down too. It doesn't matter what the number is, as long as you know which number refers to which particular group of words.



Now alter the program. The routine that deals with verbs is from Line 11Ø to 15Ø. The verbs and their numbers are entered as data in Lines 14Ø and 15Ø, set out as pairs with the numbers following the verbs.

Don't forget to re-DIMension the arrays in Line 12Ø and to adjust the FOR ... NEXT loop in Line 13Ø according to the total number of verbs you wish to use.

## VERB ROUTINES

Each of the separate verb categories—numbers—will need a separate routine.

It's difficult to give explicit instructions on how to write these routines, because a good proportion of the routines in any adventure will not be of any use elsewhere.

But there are some routines which can be used in any adventure, such as the GET and DROP routines. These can be used unchanged in any adventure that you write unless it's something very innovative. Similarly, INVENTORY—Lines 1Ø7Ø to 113Ø—is the same in any adventure, so you can use the routine with no alterations as long as the array is the same, and NB—the number of objects—has the same meaning in the new adventure.

Other routines which might find another home would include the lamp lighting routine as lighting lamps is quite a common occupation in adventures. The routine is at Lines 149Ø to 153Ø.

The remaining routines are probably not general enough for large-scale poaching, but when you write your own verb routines there are a few points to bear in mind. The routines are basically there to check if the adventurer is trying to do something to the correct object, and in the right location. If the location is wrong then the program should display a message that's appropriate to the situation, such as NOT HERE. Whatever the outcome, make sure that the adventurer knows what the effect of the last instruction was—in other words, whatever anyone tells the machine to do, there must be a printed response on the screen.

When you have worked out your verb routines enter them in the program. With the program numbered similarly to the *INPUT* adventure, the place for these routines is between Lines 1Ø7Ø and 2999.

The computer has to be able to select the correct routine according to the verb that the adventurer has used. In order that the computer can do this Line 51Ø will have to be altered.

All you have to do is to look at your verb numbers list. Now, using that numerical order, enter the start lines of the routine for each verb after the ON ... GOTO.

## HELP ROUTINE

The final routine that you should turn your attention to is the HELP routine. Consider where the adventure might need a hint, and use an IF ... THEN line to give the hint.

Other odds and ends such as the line which makes the tax inspector appear—Line 32Ø—may need to be altered or deleted according to the demands of your adventure. Also attend to the start location set in Line 28Ø.

## VARIABLES ETC.

So that you can 'get inside' the adventure program, here's a list of the variables and arrays and what they're used for.

| | |
|---|---|
| R$() | array containing the verbs and responses. |
| R() | array of response numbers. |

Corresponding elements in the two arrays above are the pairs of verbs and meanings.

| | |
|---|---|
| OB() | array containing the location number of each object. |
| OB$() | array containing the short object descriptions. |
| SI$() | array containing the long object descriptions. |

Corresponding elements in the arrays above contain information about a single object.

| | |
|---|---|
| NB | the number of objects in the adventure. Used to DIMension arrays and set up FOR ... NEXT loops. |
| L | current location of the adventurer. |
| LA | lamp status flag. Set at 1 when the lamp is on, and Ø when it is off. |
| TA | tax inspector flag. |
| N, E, S, W | exit directions. Set at 1 if there is an exit, and to Ø if there isn't. |
| I$ | the whole input before it's split into verbs and nouns. |
| V$ | the verb part of I$. |
| N$ | the noun part of I$. |
| I | number corresponding to a particular verb meaning. Used to pick out the correct routine—the one that handles that particular verb. |
| IN | the number of objects in the INVENTORY. |
| A$ | the answer to DO YOU WANT ANOTHER GO? |
| G | the number of the object dropped —element G in array OB. |

The Spectrum programs work a little differently from the others, owing to Spectrum BASIC's lack of ON ... GOSUB and ON ... GOTO. They are no great loss, though; the program still works just as well and isn't any more complicated to extend.

## LOCATION DESCRIPTIONS

The first step in altering the Spectrum program is to enter all the location descriptions from your grid.

The descriptions are put at the end of the program just as in the INPUT adventure. Enter all the location descriptions in order, each followed by the line containing the exit direction information—the variables N, S, E, and W refer to north, south, east and west, and the values Ø and 1 refer to no exit and an exit respectively.

Don't forget to enter the REM lines which will enable you to keep track of the location numbers which refer to the location descriptions.

## NEW WORDS

Make a list of all the things the adventurer will instruct the computer to do during the adventure.

You are interested in the verbs at this stage—verbs as far as adventures are concerned are not just limited to the strict grammatical definition but are either the first work in a pair, or those words that will be used on their own. Group the words together according to their meanings—the words which have the same effect on the adventure such as GET and TAKE or KILL and SHOOT. Each group will need a number, so note that down too.

The verbs and their numbers are put in the DATA statements in Lines 14Ø and 15Ø. Don't forget to enclose all the verbs with inverted commas. Follow each verb with its number according to the meanings table.

Don't forget to re-DIMension the arrays in Line 12Ø and adjust the FOR ... NEXT loop in

## Microtip

When you are planning your adventure, try to think who is going to play the game, the kind of interests and knowledge they might have—and how clever you might expect them to be. Try not to pepper the game with puzzles that would need a degree in Astrophysical Sociology to solve, for example.

If you decide to ask the player to respond with specific facts, make sure you have them correct—check tables or reference books, or you might find yourself with many disgruntled adventurers.


Line 13∅ according to the total number of words you wish to use.

The DIMensions of the array will depend on how many lines of data there are and how many items of data there are in the longest line. The first subscript is the number of pieces of data in the longest line, and the second subscript is the number of lines of data.

Feed all the data into Lines 4∅ to 7∅—if you have a lot of data create some new lines between these line numbers. In any case, count how many pieces of data are in the longest line. In the unlikely case that all the lines are the same length, you do not have to take any action. If they are not all the same length you will have to fill up all the shorter lines with zeros, to make them up to the same length as the longest or else the program will not work correctly.

## VERB ROUTINES

Each of your separate categories of verb will need a separate routine.

It is difficult to give explicit instructions on how to write these routines because a large proportion of them will be only applicable to one particular adventure. However, the *INPUT* adventure has a number of routines which will be useful as they stand in most, if not all, adventures.

For example, there are GET and DROP routines which can be copied straight into any adventure because they are fundamental routines if you are to use any objects in the game. INVENTORY will be suitable for use in most games, too; it's up to you whether you incorporate such a facility in the game, but it would be a rather strange adventure without it!

The rest of the routines depend on the requirements of the adventure. Take your lead from how the routines are structured for the rest of the existing program. Bear in mind where and to what a certain verb could apply.

Check that the instruction has been applied to the correct location and to the correct object. Try to think through which wrong instructions the adventurer could give, and structure the routines accordingly.

Once you have written all the routines you should enter them between Lines 139∅ and 2999.

The computer has to be able to select these routines according to the verb the adventurer has used.

## ARRAY G

In the Spectrum program the location description and verb routine lines are stored in the array G.

The next stage in writing your new program is to feed all of the line numbers into G. Lines

4Ø to 7Ø contain the line numbers. The first three lines are the location descriptions, and the last line contains all the starting lines for the verb routines.

With G filled with line numbers, Lines 33Ø to 35Ø will pick out the correct location description by picking out the correct element in the array. The second subscript corresponds to the array row that the number is in, so you should make sure that the subscript is correct in each of your GOTO lines, especially if you have added some extra data lines.

## MOVING AROUND

If the grid for your adventure is a different size from that used for the *INPUT* adventure you'll need to alter the movement routine at Lines 1ØØØ to 1Ø4Ø.

More specifically, the north and south lines—Lines 1Ø1Ø and 1Ø3Ø—will have to be altered if the grid is no longer six squares wide. Simply count how many squares there are across the top of your new grid and change the number 6 to the width of the grid.

## THE OBJECTS

You will have to make quite extensive changes to Lines 16Ø to 26Ø because the objects in your new adventure will almost certainly be very different from those in the *INPUT* adventure.

Count how many objects there are, and the length of both the longest short description and the longest long description. The number of objects should be your first piece of data in Line 2ØØ and will be used as one subscript when DIMensioning the arrays in Line 18Ø, and to set up FOR . . . NEXT loops elsewhere in the program. The second subscript in array B$ is the length of the longest short description, and the second subscript in S$ is the length of the longest long description.

It's clearest to use a separate DATA line for each object, but if you've written a game needing a large number of objects, you may find that it's better to have more than one object per line. Whichever way you decide to enter the data, the order must be correct, as each of the three pieces of data are fed into different arrays. The order, then, is: location number, short description, long description. If the object only appears later on in the adventure, perhaps after the adventurer has found it, or it is something which appears randomly, like the tax inspector, the location number should be zero.

## HELP ROUTINE

The final routine that you should turn your attention to is the HELP routine. Consider where the adventurer might need a hint, and use an IF . . . THEN line to give the hint.

Other odds and ends such as the Line which makes the tax inspector appear—Line 32Ø—may have to be altered or deleted according to the demands of your adventure. Also attend to the start location in Line 28Ø.

## VARIABLES ETC.

So that you can 'get inside' the adventure program, here's a list of the variables and arrays and what they're used for.

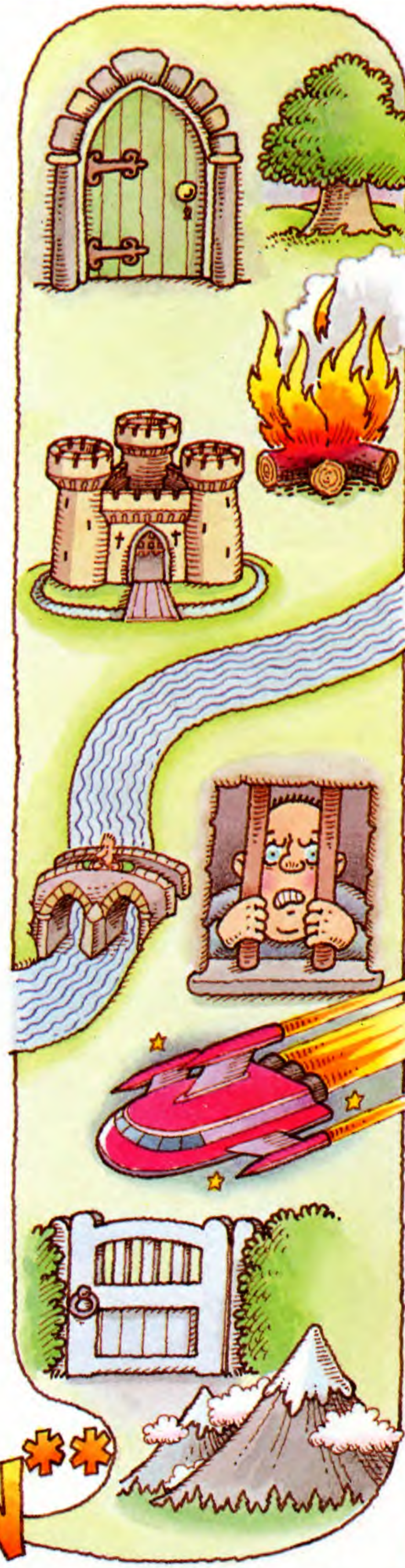
| | |
|---|---|
| G() | array containing the line numbers of the location descriptions and the verb routines. |
| R$() | array containing verbs and responses. |
| R() | array containing verb numbers. |

Corresponding elements in the two arrays above are the pairs of verbs and meanings.

| | |
|---|---|
| B() | array containing the location of each object. |
| B$() | array containing the short object descriptions. |
| S$() | array containing the long object descriptions. |

Corresponding elements in the arrays above contain information about a single object.

| | |
|---|---|
| NB | the number of objects in the adventure. Used to DIMension arrays and set up FOR . . . NEXT loops. |
| L | current location of the adventurer. |
| LA | lamp status flag. Set at 1 when the lamp is on, and Ø when it is off. |
| N, S, E, W | exit directions. Set at 1 if there is an exit, and to Ø if there isn't. |
| I$ | the whole input before it's split into verbs and nouns. |
| V$ | the verb part of I$. |
| N$ | the noun part of I$. |
| I | number corresponding to a particular verb meaning. Used to check if a particular verb has been used at some stages during the program. |
| IN | the number of objects in the INVENTORY. |
| A$ | the answer to DO YOU WANT ANOTHER GO? |
| G | the number of the object dropped—G is the element in array B. |



# REM**LOCATION**

# MOVING PICTURES-VIC/ ZX 81

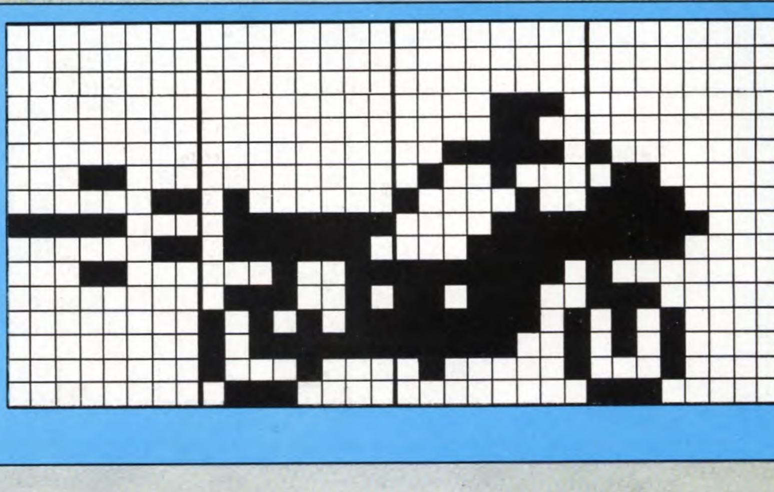
Easy machine code graphics on the Spectrum, Commodore 64, BBC Micro, Electron, Dragon and Tandy have been covered in earlier chapters. Now it is the turn of Vic 20 and ZX81 owners.

Neither of these two computers have the sophisticated graphics facilities of the others, and in fact, they cannot really be considered graphics computers at all. But there are ways of using machine code to generate simple but impressive on-screen effects.

The Vic 20 does not use sprite graphics like the Commodore 64, nor do you have to define a grid before you start building UDGs. Vic graphics are done by redefining the character set.

Each number, letter of the alphabet and symbol that appears on the keyboard appears on the screen as a pattern of pixels. These are normally laid out in 8 × 8 squares. Those pixels which are 'on' (lit on the screen) form the pattern which makes up the symbol. The rest—the pixels which are off, or not alight—make up the background.

To build up some design of your own, all you have to do is redefine which pixels are alight in a particular letter to form part of the design, then build up the whole figure by PRINTing these redefined characters next to each other. The number of characters you



**Here's a chance for Vic 20 and ZX81 owners to create some simple animated graphics. Use the routines as they are or follow the instructions to create your own characters**

- CREATING A BIKE AND SUBMARINE ON THE VIC
- MOVING THE CHARACTERS
- A FLASHING ALIEN ON THE ZX81
- PSEUDO HI-RES GRAPHICS

need is governed by the complexity of the whole image, and how much detail you can get into each character.

When building up a figure like the motorbike stunt rider shown below, you could divide the whole figure up into eight small $8 \times 8$ squares and build the pattern up that way. But the Vic 20 has a facility that will give you double-height characters. Using this facility, you only need to redefine four $8 \times 16$ characters for each figure.

Each normal-sized character is defined by eight numbers, one for each row of pixels. These numbers are worked out in exactly the same way as the numbers defining the UDGs on the other home computers (see page 38). But 16 numbers are needed to define a double-height character, again one for each row. These are still worked out in exactly the same way.

The numbers are READ into a protected area of the Vic's memory and stored there as machine code, then they are manipulated by lines of BASIC to move the figure around.

The following program creates and moves the stunt rider around the screen:

```
10 DATA 0,0,0,0,0,0,24,3,248,3,24,0,0,
   0,0,3
12 DATA 0,0,0,0,0,0,0,64,127,127,19,
   114,171,191,136,112
14 DATA 0,0,0,14,12,62,72,132,15,31,
   254,220,253,249,65,0
16 DATA 0,0,0,0,0,128,96,240,248,240,
   128,224,80,80,16,224
18 DATA 0,0,0,0,0,27,0,13,48,198,0,0,
   0,0,0
20 DATA 0,0,0,0,0,0,0,71,63,63,4,26,
   43,37,18,12
22 DATA 28,24,60,80,79,143,143,30,
   246,190,252,240,128,0,0,0
24 DATA 128,96,248,240,224,192,112,
   168,168,136,112,0,0,0,0,0
100 FOR Z=0 TO 15+7*16:READX:
    POKE 6144+Z,X:NEXT
105 FOR Z=0 TO 15:POKE 6656+Z,
    0:NEXT
110 POKE 36867,255:POKE 36869,
    254:A$="■ABC":B$="■EFG":
    C$="@D"
120 POKE 36879,25:POKE 36878,15
125 PRINT "■"
200 FOR Z=0 TO 255:GET Z$
210 PRINT "■"TAB(Z)"□";:POKE
    646,RND(1)*3+2:PRINTMID$
    (C$,RND(1)*2+1,1);
220 IF Z$="" THEN PRINT A$:POKE
    36877,200+RND(1)*5:S=S+20
230 IF Z$="■" AND Z<230 THEN
    PRINT"■";:Z=Z+22:PRINT
    TAB(Z)"□■@"A$:GOTO 260
240 IF Z$="■" AND Z>30 THEN
    PRINT"■";:Z=Z-22:PRINT
    TAB(Z)"□■D"B$:GOTO 260
250 IF Z$<>"" THEN PRINT B$:POKE
    36877,240+RND(1)*5:IF S>0
    THEN S=S-40
260 FOR C=1 TO S:NEXT C,Z:GOTO 125
```

Lines 10 to 24 contain the DATA which defines the shape of the image. Each line defines a different character so there are 16 DATA entries in each one. If you want to change the image you can alter the DATA here to redefine each character and so what is being displayed on the screen.

Line 100 READs the DATA and POKEs it into memory locations 6,144 to 6,271. Line 105 fills the space character with zeros to ensure that you have a blanking out routine where nothing is PRINTed on the background.

In Line 110, POKEing 36867 with 255 switches on the double-height character facility, and POKEing 36869 with 254 redirects the start of character memory pointer to ensure that the particular character set you have redefined is being used. (There are two other character sets starting at 5,120 and 7,168 which can be accessed by POKEing 253 and 255 respectively into 36869.)

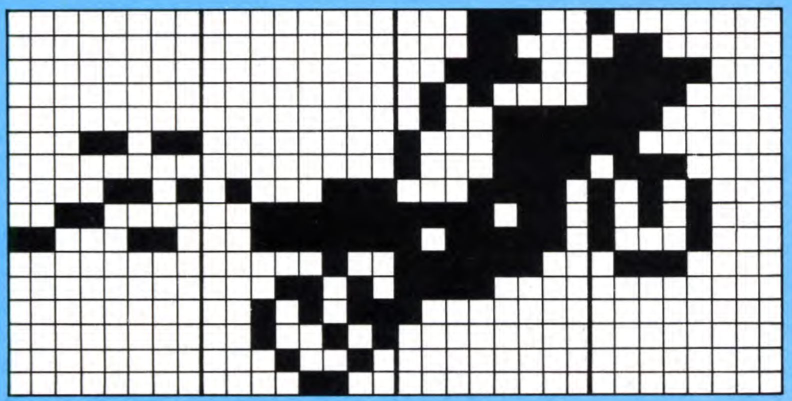
A$ and B$ put the characters ABC and EFG together to make up the two figures of the stunt rider—one driving normally, the other doing a wheelie, as shown in below. The reverse arrow at the front end of these strings tells the Vic to make the motorbike blue. The string C$ contains the two flame characters.

On Line 120, POKEing 36879 with 25 turns the screen border white, and POKEing 36878 with 15 sets the sound volume. Line 125 clears the screen. Line 200 sets up the FOR ... NEXT loop which moves the motorbike across the screen, while the GET$ readies the machine for the keypresses which control the bike's movement.

PRINTing a reverse S at the beginning of Line 210 homes the cursor, which prevents the screen scrolling. POKEing 646 changes the colour of the next character—in this case randomly—and the rest of the line PRINTs randomly either of the two flame characters.

The next five lines PRINT and move the bike along. Press the left/right cursor key and the bike will travel down the screen, while pressing the up/down cursor key and the bike will do a wheelie and move up the screen. Pressing any other key makes it speed up, move forward and do a wheelie.

The reverse £ sign makes the flame red when moving up or down, or a random colour otherwise. POKEing 36877 produces the sound effects. And the variable S controls how fast the computer goes round the loop, which in turn controls how fast the bike moves across the screen.



## THE SUBMARINE

To construct the submarine, the following program only needs half as much data. The graphic is only eight pixels deep, so you do not need to use double-height characters. Again the data are fed in a character at a time, but this time you only need eight numbers to define the character. The ten items of data in Line 5 are used to create the random white wave dot pattern.

```
5 FOR Z=Ø TO 9:READK(Z):NEXT:
   DATA Ø,128,64,32,16,8,4,2,1,Ø
1Ø DATA Ø,Ø,Ø,Ø,127,255,127,63
12 DATA 4,7,31,31,255,255,255,255
14 DATA Ø,Ø,Ø,Ø,254,237,254,252
16 DATA Ø,Ø,128,Ø,62,62,Ø,128
18 DATA Ø,Ø,1,Ø,124,124,Ø,1
2Ø DATA Ø,Ø,Ø,Ø,127,233,127,63
22 DATA 32,224,248,248,255,255,
   255,255
24 DATA Ø,Ø,Ø,Ø,254,255,254,252
1ØØ FOR Z=Ø TO 7+7*8:READX:
   POKE 6144+Z,X:NEXT
1Ø5 FOR Z=Ø TO 7:POKE 64ØØ+Z,Ø:
   NEXT:POKE 65Ø,128
11Ø POKE 36867,24:POKE36865,6Ø:
   POKE 36869,254:A$=
   "■□π@AB■□":B$=
   "■□πEFG■□"
12Ø POKE 36879,1Ø9:POKE 36878,15:
   C$=A$
125 Z=12Ø:PRINT"♥"TAB(Z);C$:G=Ø
2ØØ GET Z$
21Ø PRINT "▤"TAB(Z);
22Ø IF Z$="." THEN C$=A$:Z=Z+1:
   IF Z>126 THEN Z=126:PRINT
   "♥"TAB(Z);
23Ø IF Z$="," THEN C$=B$:Z=Z-1:
   IF Z<111 THEN Z=111:PRINT
   "♥"TAB(Z);
24Ø IF Z$="□" AND G=Ø THEN G=1:
   N=Ø:GOTO 3ØØ
245 W=K(RND(1)*1Ø)
25Ø POKE 64ØØ+INT(RND(1)*8+1)*2,W
26Ø PRINT C$
27Ø IF G=1 THEN 32Ø
28Ø GOTO 2ØØ
3ØØ IF C$=A$ THEN X=Z+4:CH=3:ZZ=1
31Ø IF C$=B$ THEN X=Z:CH=4:ZZ=-1
315 GOTO 335
32Ø N=N+1:POKE 768Ø+X,32:POKE
   384ØØ+X,1:POKE 36875,128+
   N*3:X=X+ZZ
325 IF (X<11Ø OR X>131)THEN POKE
   36875,Ø:G=Ø
33Ø IF G=Ø THENFOR D=ØTO1Ø9:
   POKE 36877,D+13Ø:POKE 36879,
   D:POKE36877,23Ø-D:NEXT:
   GOTO 2ØØ
335 POKE 384ØØ+X,2:POKE 768Ø
   +X,CH
34Ø GOTO 2ØØ
```

The program is much the same as the stunt bike's. In this case it is the full stop and the comma key which turn the submarine around by changing frames, and move it in the opposite direction. The POKE 65Ø,128 on Line 1Ø5 gives the effect of repeating a key press when you hold the key down. And Lines 245 and 25Ø give the random pattern of white wave dots on the blue background.

The new thing here is the torpedo firing mechanism. When the space bar is pushed, the computer jumps to the fire routine. Lines 3ØØ and 31Ø look at which way the submarine is facing and decide which way round the torpedo should be, where it should start from, and which way it should run.

Line 32Ø moves the torpedo across the screen. The POKE 768Ø+X,32 rubs out behind the torpedo by overprinting the old image with a space. The POKE 384ØØ+X,1 then POKEs the background colour into the space. And POKE 36875 gives you the sound.

Line 322 checks to see when the torpedo has reached the edge of the screen area, turns off the sound again and cancels the torpedo graphic by setting G to Ø. When the torpedo hits the border, Line 33Ø gives different sounds by POKEing 36877 with a succession of different values, and flashes the border by POKEing 36879.

Line 335 just POKEs the torpedo graphic onto the screen with POKE 768Ø+X,CH and colours it red by POKEing 384ØØ+X with 2.

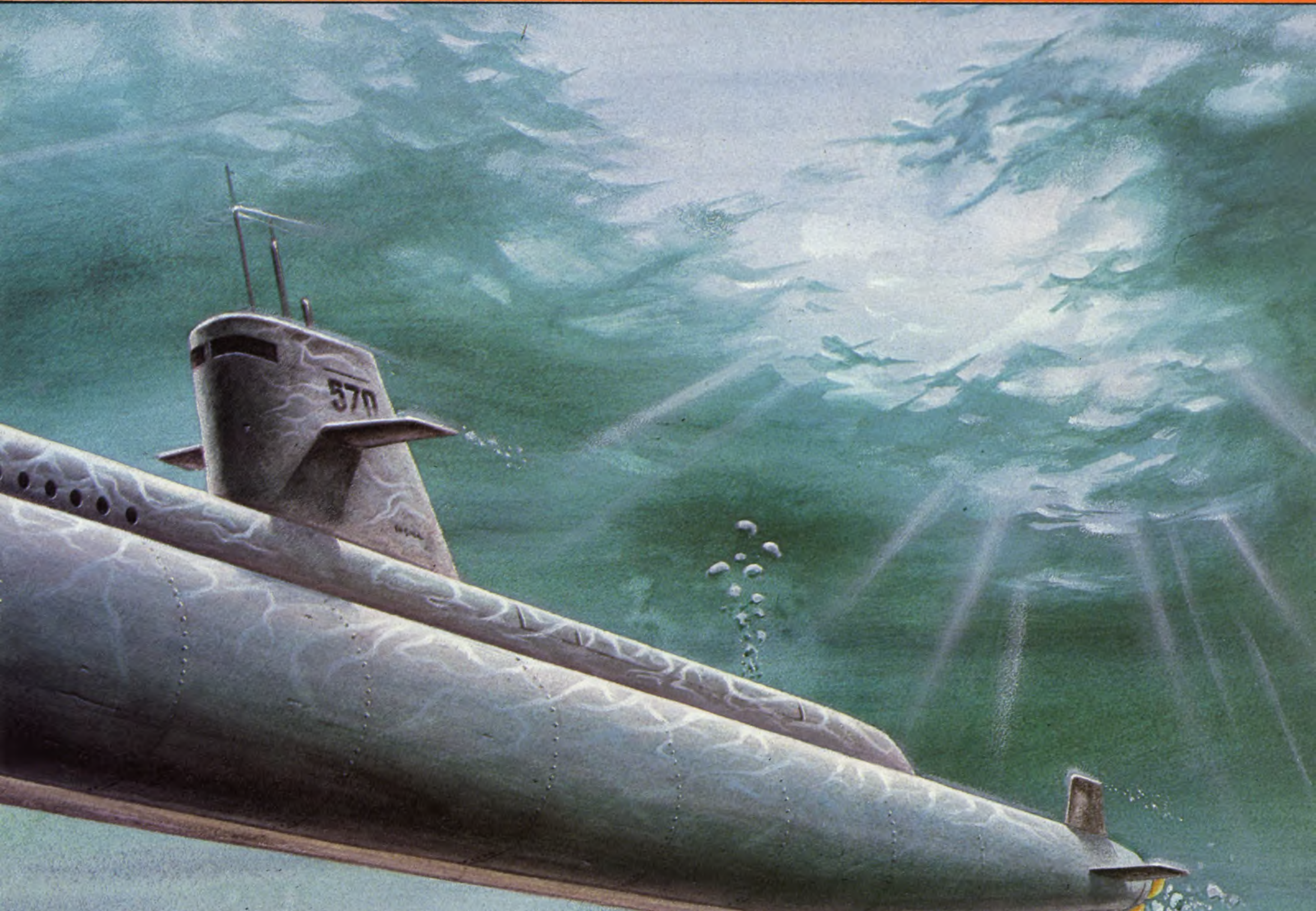
The only way to put graphics on the screen with the ZX81 is to PRINT the graphic symbols shown on the keyboard. But machine code can be useful here too.

The following program creates an alien:

```
1 REM ..........................................
   ........................................
1Ø LET A$="2AØC4Ø233ABC4Ø471121ØØ
   FEØØ28Ø319"
15 LET A$=A$+"1ØFD3ABD4Ø16ØØ5
   F1911BE4Ø3ABB4ØFE"
2Ø LET A$=A$+"ØØ28Ø311CE4ØØ6Ø4
   C5Ø6Ø41A7723131Ø"
25 LET A$=A$+"FAØ11DØØØ9C11Ø
   FØC9Ø1ØØØØ"
3Ø LET A$=A$+"ØØØØØØØØØØØØØØØØ
   ØØØØØØØØØØØØØØØØ"
35 LET A$=A$+"878B8BØ48Ø85Ø58Ø
   Ø28Ø8ØØ1Ø685Ø586"
5Ø FOR N=16514 TO 166Ø5
6Ø POKE N,16*CODE A$+CODE A$(2)
   -476
7Ø LET A$=A$(3 TO )
8Ø NEXT N
```

Line 1 must contain at least 92 characters to accommodate the 92 bytes of the machine code program, which are entered in this BASIC program as the character string A$.

Lines 1Ø to 25 contain the machine code which actually prints the alien on the screen. And the Øs—16 bytes of them in all—in Line 3Ø overprint the graphic with blank spaces when you move it.

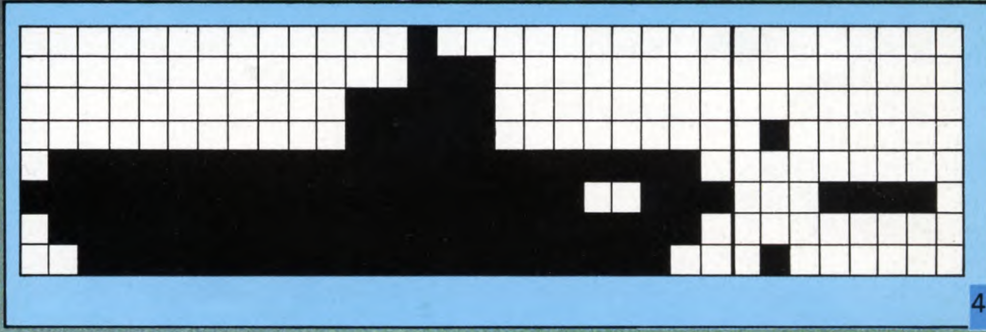


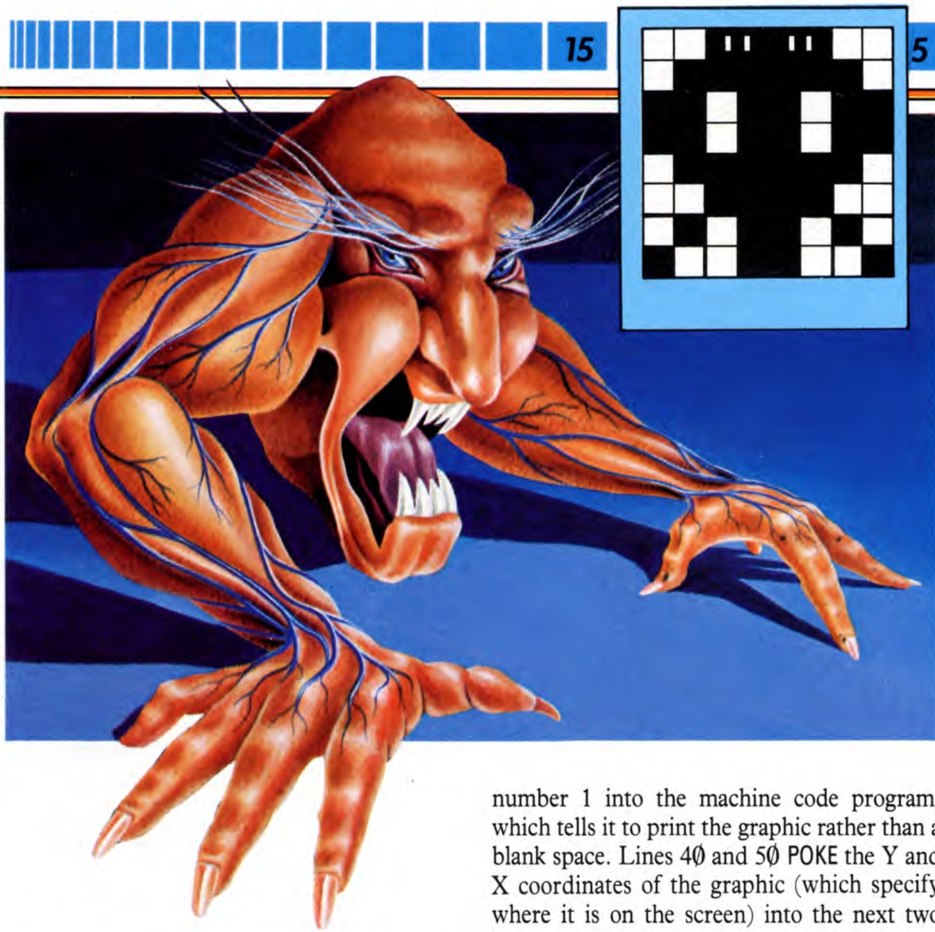
Line 35 contains the details of the alien itself. If you look at the string byte-by-byte—a pair of digits at a time—and compare them with the character set in Appendix A of your ZX81 manual, you will see that they correspond to the graphic and inverse characters.

These characters build up the alien block-by-block in an 8 × 8 square, starting at the top left and working across and down to the bottom right. As the graphics characters themselves break the character square into quarters, this gives an effective resolution of 8 × 8.

You can alter the graphic by changing the numbers in this line. Be careful not to use the code for any character that takes up more than one character space—like AT, TAB, THEN or LEN. These will crash the program when you call it. Try designing your own alien on an 8 × 8 grid. Then put the appropriate numbers into Line 35.

When you RUN this program, Lines 5Ø to 8Ø POKE the data in the strings into the space occupied by the REM statement. And if you LIST the program again you will see the symbols corresponding to the machine code





appear in the REM statement. The last part of it will show you the graphics characters that build up the alien, end to end.

Now that the machine code has been POKEd in and is protected by the REM statement, you can delete all of this program except the line containing the REM statement itself. Then you can key in the following program which will make the graphic move around the screen:

```
1Ø LET X=14
2Ø LET Y=8
3Ø POKE 16571,1
4Ø POKE 16572,Y
5Ø POKE 16573,X
6Ø RAND USR 16514
1ØØ LET A$=INKEY$
11Ø IF A$=""THEN GOTO 1ØØ
12Ø IF A$="Z" AND X>Ø THEN LET
   X=X-1
13Ø IF A$="X" AND X<28 THEN LET
   X=X+1
14Ø IF A$="P" AND Y>Ø THEN LET
   Y=Y-1
15Ø IF A$="L" AND Y<2Ø THEN LET
   Y=Y+1
16Ø POKE 16571,Ø
17Ø RAND USR 16514
18Ø GOTO 3Ø
```

Lines 1Ø and 2Ø start the alien off in the middle of the screen. Line 3Ø POKEs the number 1 into the machine code program, which tells it to print the graphic rather than a blank space. Lines 4Ø and 5Ø POKE the Y and X coordinates of the graphic (which specify where it is on the screen) into the next two bytes of the program. The program is then called and the graphic is printed on the screen.

Lines 1ØØ to 15Ø contain the standard routine to move something around the screen (see page 57). When the program is RUN, pressing the Z key will move the alien left, pressing X moves it right, P up and L down.

Line 16Ø POKEs Ø into the machine code program, and when Line 17Ø calls the program again it tells the computer to print blank spaces instead of the graphic. This is to rub out the graphic so that it can move without leaving a trail on the screen. Line 18Ø sends the computer back round the circuit so that the graphic can be reprinted in its new position.

## INVERSING THE SCREEN

There are several other useful things you can do with the ZX81 screen with machine code. The following program inverses the screen—that is, it changes everything that is black into white and everything that is white into black. It can be fed in using your machine code monitor (page 280), although the routine should be called from within a BASIC program:

```
2A ØC 4Ø Ø6 17 23 7E FE 76 28 Ø5 C6 8Ø 77
18 F5 1Ø F3 C9
```

Call this with the command RAND USR start address. The direct command RAND USR, without a line number, clears the screen first so the effect of the routine is not terribly impressive. But even a simple program like:

```
2Ø LIST
3Ø RAND USR
```

will do, where RAND USR is followed by the start address of the machine code routine.

You could, of course, add this routine to the alien program above. The line of machine code data should be added to the first program, the one that creates the alien, as part of A$, in a line like:

```
4Ø LET A$=A$+"2AØC4ØØ617237
   EFE7628Ø5C68Ø7718F51ØF3C9"
```

Note that this time the bytes are closed up with no space between them.

Line 5Ø must be altered to POKE the extra data into the REM statement and should read:

```
5Ø LET N=16514 TO 16624
```

And the REM statement in Line 1 should also be lengthened by 19 characters. When the first program has been RUN and all the lines except the first REM statement have been deleted, you have to modify the second program so that it will call the inverse routine. This is done simply by adding:

```
155 IF A$="I" THEN RAND USR 166Ø6
```

The memory location 166Ø6 is where this new piece of programming starts, and that routine will be called when you press the I key. Press the I key once, and the alien will change from black to white and its background will go from white to black. Press it again, and it will change back. Hold the I key down to make it flash.

## PSEUDO HI-RES GRAPHICS

As you no doubt know, the ZX81 will not display high resolution graphics. But there is a way of producing what look like hi-res graphics using machine code. This simple program POKEs numbers into the first five memory locations in a REM statement, then calls the machine code program formed that way.

```
1Ø REM.....
2Ø POKE 16514,62
3Ø POKE 16516,237
4Ø POKE 16517,71
5Ø POKE 16518,2Ø1
6Ø FOR N=Ø TO 3Ø
7Ø POKE 16515,N
8Ø RAND USR 16514
9Ø NEXT N
```

This will give you something on the screen that looks very much like hi-res graphics. Unfortunately, there is no easy way you can move or control them.

# IMPROVING YOUR DISPLAYS

- PLANNING THE DISPLAY
- HOW TO POSITION TEXT ON THE SCREEN
- IMPROVING THE LAYOUT
- ADDING COLOUR

**The title page and other screen displays in your program need to be carefully planned and constructed if you want a really professional look. Here's how to do it**

From the user's point of view, there are several things that make the difference between a program which seems well-designed and easy to use and one which does not. Of course, the first thing which will make the difference between a professional-looking program and an amateurish one is that it must work properly, without any bugs or awkward user routines. And at a more theoretical level, the program itself should be well structured and easy to follow. The techniques you need to ensure good structure and freedom from errors have already been covered, on pages 173, 217, 334 and 375.

But even the best-written program is going to look sloppy and amateurish if it isn't well presented—in particular, if the screen display is not neat and easy to follow. This is fairly simple to ensure, but it does mean that you need to give careful thought to formatting the display. This means positioning PRINT and INPUT statements correctly to create a neat and interesting layout on the screen.

The article on page 117 explained the various BASIC commands available on your computer to control the position of characters on the screen. Now, you will see how to use them to set up a title page for an imaginary game, 'INPUT'. Similar techniques can also be used for displaying instructions, a menu, or a prompt, and the first thing to do is to look at tidying these up.

## CLEAR INSTRUCTIONS

Nothing looks sloppier than a screen display which is mis-spelt or which hasn't been thought out properly. For example, you may have noticed how many otherwise very good games make such mistakes as PRINTing 'you have one lives left'.

This particular case is easily put right with a simple IF ... THEN statement (of the form: IF L=1 THEN PRINT "life"), and it reflects badly on the programmer for not changing it. Make sure that you remove any bugs of this type from your programs before you start to worry about the format.

You will probably find it helpful to keep a few guidelines in mind when you are creating a screen display.

The most important thing to aim for is clarity. If words are too close together then they will be hard to read, so space out the different lines carefully. Make sure, too, that none of the words are split onto two lines. If this cannot be avoided, you should put a hyphen at the end of the first half of the word and make the break at a logical point.

If you need to PRINT something which is INPUTted by the user, such as the name on a 'hall of fame', you must ensure that what is PRINTed does not interfere with the rest of the screen. To do this, you could include a routine which either doesn't allow names of more than a certain number of letters (see page 377 for an error-trapping procedure to do this), or which makes sure that, whatever the length of the name, it is put in a position that does not affect the position of the score or other PRINTed information on the screen.

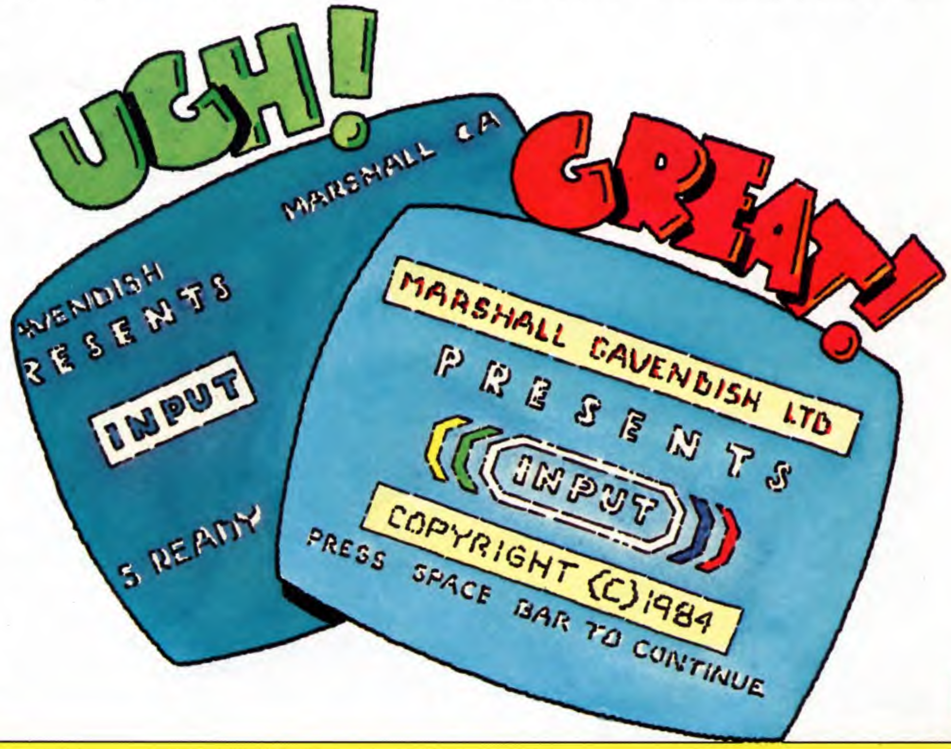
Where you have a list, again such as a high score table, you should PRINT each line starting at the same column. This may seem obvious, but even some commercial programs do not do this.

The second thing to bear in mind when you are designing a display, is that you should not try to cram too much information onto the screen at once. If you do put too much on, the program user will not be able to remember it all, and so will not be able to make the most of your program.

At the same time, do not put so little information onto the screen that you need to PRINT a large number of screens. The hard part is to get the balance right. Of course just what this balance is depends on the program—the simpler your program is, the better the user will be able to understand it, and the less instructions you will have to display.

## USING COLOUR

Try and make the screen interesting. If you can have a variety of colours on the screen at once, do so: it brightens up the display, and so makes it easier for the user to read, or to pick



out particular elements. You can choose whether to colour the text or the background or both, or just have a coloured border.

The Spectrum is lucky here, in that it has a very versatile FLASH command. The changing picture that results from using flashing colours (or from changing the colours of certain words or objects if your computer does not have a FLASH command or flashing colours) provides a form of motion to your picture, which is more interesting to watch than a simple static display.

## POSITIONING

As well as trying to follow these basic guidelines, you should always position words on the screen so that they look as if they are where they should be.

For example, type in the following short program and RUN it. This is how NOT to make a title screen for your program. The words are all over the place: none are co-ordinated with any other words, and so the result is a display which looks a mess. Later on, you'll see how to sort this out:

```
10 PRINT "here is a new game caled input"
20 PRINT AT 5,17; "©1984"
30 PRINT AT 12,13;"□by
   MarshallCavendish"
40 PRINT AT 18,25;"any key"
50 PAUSE 100
60 CLS: STOP
```

```
10 PRINT TAB(27)"MARSHALLCAVENDISH"
   TAB(99)"PRESENTS□"
20 PRINT "■■"TAB(10)"■INPUT"
30 PRINT"■■COPYRIGHT(C)
   □1□9□8□4"
40 FOR Z=1 TO 2000:NEXT Z:
   PRINT "□"
```

```
5 CLS
10 PRINTTAB(20,20)"INPut BY
   marSHALLcaVENDISH":PRINT"(c)1984"
20 FORT=1 TO 2000:NEXT:CLS
```

```
10 CLS4
20 PRINT@40,"INPUT"
30 PRINT@136,"copyright
   marshallcavendish";
40 PRINT@228,"PRESENTS";
50 FOR T=1 TO 2000: NEXT:CLS
```

You can see how unimpressive this display is. There are several things which ought to be improved.

Firstly, capitals could be used to good effect: the Spectrum version uses almost entirely lower case letters, even for the first word of the display, and for the name of the game. Capitals can be used to stress important words, or even for all the words, in order to give the display a more professional appearance.

Secondly, the screen should always be cleared before PRINTing a new message. Otherwise, as with the program above, the remainder of anything already on the screen stays in sight, making the display untidy.

There is no space in between the two words Marshall and Cavendish. There should be, and it looks especially untidy with the words PRINTed as they are now—with 'Cavendish' overflowing onto the next line. See below to find out how you can position your PRINTs correctly.

The program does not wait for you to press a key before continuing, so that you have not really got enough time to read the screen properly before it is wiped off. This should not happen, and there ought to be a line of text telling you what to do when you want to continue.

You can work out where you want to PRINT things on the screen quite easily, from the dimensions of your computer's screen, and the number of characters in the words.

For example, suppose your computer has a screen with 22 lines of 32 characters, and you want to PRINT a phrase 10 characters long in the middle of a line one line from the top of the screen. When you work out the length of your phrase, don't forget to include all the spaces between words.

To work out the horizontal coordinate of the start position of your phrase, subtract the length of the phrase (here 10) from the total number of characters in the line (here 32), and divide the result by 2. You divide the result by two so that you get an equal number of spaces either side of the PRINTed phrase.

In this example, the answer is 11. Because 0 is counted as a number, the first character space is numbered 0. So subtract one from the answer above, 11, to get the horizontal PRINT AT position (or the PRINT TAB position).

Sometimes, you may wish to PRINT something only about two or three characters in from the left hand side of the screen. In this case, it is probably easier to include that number of spaces inside the PRINT statement, than to use a PRINT TAB or AT statement. But don't do this if you need to use lots of spaces, as it tends to be very wasteful of memory.

You can work out the vertical position in the same way as you did the horizontal: if you want the phrase in the middle line of the screen, work out how many lines the message will cover. Subtract this number from the total number of lines in the screen, and divide the answer by two. As before, subtract one from this (to compensate for the fact that the computer counts 0 as a number) and the result is the line number at which you want to PRINT.

If you want to PRINT at a position other than the middle of the screen, you can either estimate what you think the coordinates are or you can use graph paper and work out the exact numbers.

Spectrum owners should remember that the Spectrum's PRINT AT command works slightly differently to other computers'. The difference is that the Spectrum uses the first number after the PRINT AT command as the vertical y coordinate, and the second one as the horizontal x coordinate. When PLOTting, though, the x coordinate comes first, as is more usual.

With both computers, the position Ø,Ø is at the top left-hand corner when PRINTing, but at the bottom left-hand corner when PLOTting and DRAWing.

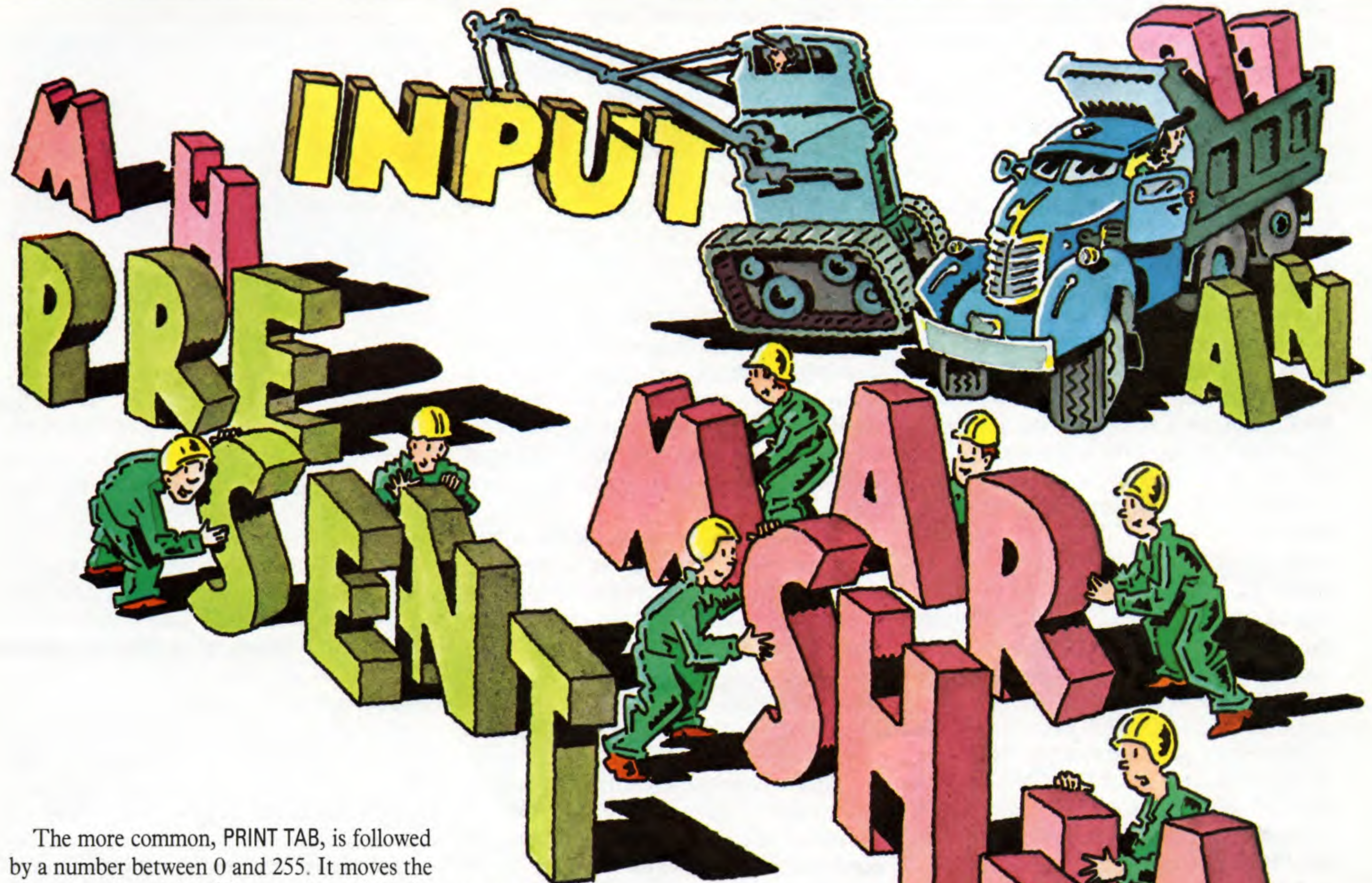
Commodore BASIC has no PRINT AT command, but lets you move the cursor position so that the next PRINT position comes where you want it to.

This means that you do not have to work out the numbers to go after a PRINT AT statement—you just include however many cursor ups/downs/lefts/rights in your PRINT statements as you want.

If you are trying to position the PRINT statement so that it appears in the middle of the line, follow this simple procedure: subtract the length of the statement from 40 for the 64, or 22 for the Vic (there are 40 characters in a line on the 64, and 22 in a line on the Vic) and divide the answer by two. The number you end up with is the number of spaces there should be either side of the PRINTed statement on your screen.

You simply put this number of cursor rights at the beginning of your PRINT statement, and the word or words will appear in the correct position in the line.

The different symbols for cursor up, down or whatever, can be very confusing. So have a look at pages 420 and 421 for more information on what the symbols look like and what they do. The Commodore does have a more limited alternative: you can use PRINT SPC or PRINT TAB. Both of these commands refer solely to the horizontal position of the PRINT statement, although by using numbers larger than 40 after them you can get limited vertical control as well.




The more common, PRINT TAB, is followed by a number between 0 and 255. It moves the cursor to the right. The number after the command dictates how many character squares to the right the cursor moves. In fact, the computer counts every PRINT TAB from the beginning of the line, rather than from the current cursor position.

PRINT SPC is very similar. The only difference, in fact, is that PRINT SPC counts its 'cursor rights' from the current cursor position.

If you want to position the PRINTed words so that they are in the centre of the screen, as before, you follow the same set of calculations and use the resulting number as the number in the brackets after the PRINT TAB or SPC.

You might expect these two types of PRINT statements, especially the PRINT SPC, to PRINT the relevant number of spaces, but in fact this is not the case: they just move the cursor.

As with many commands on the Commodore, you can achieve the same effects with certain POKE commands. These two POKEs move the cursor to the specified position:

```
POKE 781, (desired horizontal position)
POKE 782, (desired vertical position)
```

If you use these two POKEs to move the cursor, you should place this command in front of every PRINT statement which uses them:

```
SYS 65520
```

Following this call, PRINTing starts at positions set by the POKEs. As there are 25 lines in the Commodore's screen, POKEing the second of these POKEs with a number greater than 25 causes the computer to start at the top of the screen once it is at 25, and count down again. The same happens if you POKE the first number with a value greater than 40.

The number after the PRINT @ command, as you probably know, refers to a character square on the computer's text screen. As there are only 512 squares on the screen, this number cannot be more than 511, or an error message will appear.

You can work out the number of the character square at which you want to start PRINTing quite simply. The first thing to do is to decide which line you want to PRINT @ (remember to count from Line Ø). Then multiply this number by 32, the number of characters in a line.

Then add a number between 0 and 31 to this to find the number of the square you want.

If you want to PRINT a word (or words) so that the statement is exactly in the centre of the line, first count how many character spaces the statement will take up. Subtract this length from 32, and divide the number by two (so that the remaining space on the line will be split evenly between the two sides of the line) and add the answer to the number of the first space in the line. You'll have to round up or down any halves. The number you have left is the number of the first space that you will use in the PRINT @ statement.

Try to work out the PRINT @ numbers for a variety of screen positions and a variety of words of different lengths. You can check your answers by PRINTing @ them, and seeing if the words that appear are where you want them.

You might like to start PRINTing at a certain fraction of the line: say, half or one third of the way through the line.

This is very easy: divide 32 by the relevant number (2 for a half, 3 for a third, . . .) and add this to the number of the first space in the line, just as before.

Wherever you want to PRINT @ a position which is not exactly half way through a line, but is at some obscure position, you just have to guess roughly what the number is. Just experiment until the result looks right.

You can use a grid like the one in the user manual, and actually plan your PRINTing on that: then all you need to do is to read off the PRINT @ numbers from the scales along each side. The trouble with this is that it takes a long time and so is not really worth doing unless you have quite a few different statements to work out.

## A BETTER DISPLAY

The following programs give you a better title page, although they do not need any more complex programming techniques. They put right all that was wrong in the last program, and add colour and motion to try to make the screen a little more interesting. You should note especially the use of PRINT AT and PRINT TAB (or, for the Commodore, cursor control characters), as these are needed for almost any PRINTing that you will do.

```
5 LET X=1
10 BORDER 1: PAPER 1: INK 7: CLS
20 PRINT INVERSE 1;AT 3,3;"□MARSHALL
   CAVENDISH LTD□"
30 PRINT INVERSE 1;AT 5,10;
   "□PRESENTS:□"
40 PAUSE 50
50 PRINT PAPER 6; INK 1;AT 10,10;
   "□I□N□P□U□T□"
60 PRINT PAPER 6; INK 2; FLASH 1;
   AT 9,9;"▛▀▀▀▀▀▀▀
   ▀▀▀▀▜";AT 11,9;"▙▄▄▄
   ▄▄▄▄▄▄▄▄▟"
70 PRINT PAPER 6; INK 2; FLASH 1;AT
   10,9;"▌";AT 10,21;"▐"
80 PRINT PAPER 5; INK 0;AT 15,2;
   "COPYRIGHT AUDIO, VISUAL, 1984"
90 PRINT ''''"□□□PRESS ANY KEY TO
   CONTINUE"
100 PAUSE 0
110 CLS
120 PRINT "□□□□□INPUT PART 1
   INDEX"
130 PRINT
140 FOR X=1 TO 10
150 READ a$
155 READ b$
160 PRINT 'TAB 3;a$;TAB 25;b$
170 NEXT X
180 DATA "Animation","26-32","Basic
   programming","2-7","BREAK,
   Dragon","7","Cassettes","25",
   "Cassette recorders","24",
   "CHR$ , use of","26-27",
   "CLEAR","10-27","CLOAD,
   Dragon", "14","CLS, explanation
   of","27","CODE , Spectrum","8"
```

The first thing to notice about this Spectrum program is that it changes the colour of the screen and BORDER, and clears the screen. It does not matter what colours you choose for the screen, except that if you can avoid the normal white BORDER and PAPER with black ink it usually looks better, simply because it is different.

You should make the display as easy to read as possible, which means having an INK colour which stands out from the PAPER. Also, some colour combinations are considerably more pleasant than others.

## POSITIONING

Lines 20 and 30 PRINT the words 'MARSHALL CAVENDISH PRESENTS'. But the words are not all on the same line; the name of the company, Marshall Cavendish, is on the top line with the word 'presents' two lines beneath. This makes the text on the screen look much clearer.

The program uses PRINT AT to position these words in the centre of the screen. Try to work out what the PRINT AT numbers should be, using the calculations given above. They should be the same as those in the program.

You can see the advantages of putting these words in their correct positions when you RUN the program: unlike the previous program, the words look as if they are in the right place.

You could have used PRINT TAB to PRINT the same words by changing Line 30 to:

```
30 PRINT 'TAB 10; INVERSE 1;
   "PRESENTS: "
```

There is an apostrophe before the TAB 10 so that the Spectrum PRINTs a clear line before PRINTing 'PRESENTS'. You can also get the Spectrum to PRINT a blank line just by entering the command PRINT (with a suitable line number) on its own.

The program PAUSEs for one second after this, to give more emphasis to the next item to be PRINTed (the name, INPUT).

Lines 50, 60, and 70 PRINT "INPUT" with the FLASHing red and yellow border. The border is made up of ROM graphics, as you can see from the program, which FLASH red and yellow.

**A first attempt**

These Lines use a series of PRINT AT commands, and show how flexible this command is. You don't need to repeat the PRINT so long as you don't put a semi-colon in the Line in between the items you want to PRINT.

The various ATs are separated by semi-colons. They could equally well be separated by commas, except that a comma would cause half a line of spaces to be PRINTed, which might blank out what was on the screen already, and so it is a good idea always to use semi-colons: unless, of course, you want to blank out what was on the screen at that position.

Working out the PRINT AT coordinates for the FLASHing border cannot be done using the calculations above, since the border is not at the centre of the lines. To work out the positions, then, you must take the coordinates from the word inside the border, and try to add or subtract something to get the numbers for the border.

For example, take the top line of the border. You know that the PRINT AT coordinates of 'INPUT' are 10,10. You also know that the top line of the border is one line higher than this. So you subtract one from the first number (10) to give your first coordinate, 9.

You want the border to start one space before the word, so you also subtract one from the second number of the 'INPUT' coordinates, to give 8.

Try to work out the numbers for the other parts of the border, and check your answers against the coordinates given in the program above.

The program finishes the screen display by PRINTing a copyright message, to remind users that it is illegal to copy programs, and telling them to press any key to continue.

In fact when you do press a key, instead of a game starting, or you being presented with instructions for a game, as might happen in a game program, you are presented with a section of the INPUT part one index.

What happens when you press a key is that


**The finished display**

the Spectrum clears the screen, and PRINTs the message 'INPUT PART 1 INDEX' at the top of the screen. As you can see if you look at the program, there is no PRINT AT or TAB command, so why is the message in the middle of the line?

Since only five free character squares are needed to place the message in the middle of the line, spaces are put at the start of the PRINT statement (inside the inverted commas) instead of using a PRINT AT command. After this, the computer PRINTs a line of blank spaces: PRINT on its own does this (Line 13Ø).

The FOR ... NEXT loop which follows reads two string variables from the DATA in line 18Ø. You might think it strange that the page numbers are stored in a string variable form. The reason for this is that some of the entries have more than one number: Animation, for example, is on pages 26–32. You could store '26–32' in a numeric variable, but when the computer PRINTs it, it would treat it as a subtraction, and PRINT—6! That would really confuse any user!

On top of this, some of the entries have page numbers separated by a comma—and as you know, a comma is the punctuation mark used for separating different items in a list of DATA, so the two page numbers, although they refer to the same entry, would be treated as if they belonged to two entries.

You might also think that it would be clearer to store the DATA for the entries in a separate Line to the one used for storing the page numbers. The RESTORE command would enable the program to switch between the two separate Lines, and all would be fine. Wouldn't it?

In fact the answer is no, because the RESTORE command sends the computer to the beginning of the relevant Line of DATA, and so the computer would PRINT out the same entry and the same page number every time, which is all very well if you want to know where Animation is, but not otherwise!

Of course, one way round this is to use a separate Line for each item of DATA, but that is very inefficient.

Line 16Ø controls the PRINTing of the DATA, and sets the PRINT position. The PRINT TAB command is used here to align the entries and page numbers so that they all start in the same column, and look neat.

PRINTing an index is an ideal use for PRINT TAB, as the row is automatically changed by the computer with every different PRINT statement, and the TAB sets just the position along the line. The entries are PRINTed at the third column in from the left, which is set by TAB3. The numbers are then PRINTed in the same line, but starting 25 columns in.

Using PRINT TAB both must be PRINTed with one PRINT command: if you look at the program, Line 16Ø has just one PRINT, and separates each part with semi-colons. If a colon was used followed by another PRINT command, the numbers would be PRINTed one line below their respective entries, which would be rather confusing.

If you change the TAB 25 to TAB (25+32) or TAB 57, then there would be no difference. This is because the TAB command moves the cursor to the specified column, and if the number after TAB is more than 32 it is divided by 32 to leave the remainder. It always stays on the same line unless backspacing is needed.

Backspacing is only needed if you try to PRINT TAB (a number less than the column where the cursor is). When you do this, the Spectrum moves on to the next line of the screen, and PRINTs there.

Try changing the TAB number to get a rough idea of what various numbers actually mean in terms of screen positions. Remember that there are 32 characters in each line.

PRINT AT and PRINT TAB do very similar functions, but in different circumstances. Whenever you want to PRINT a list of words or numbers on the screen (such as an index or a high score table) you should use PRINT TAB, as this is more efficient: you do not need to repeat it, or to specify the row. PRINT AT, on the other hand, is very useful for PRINTing a small number of items, or if you want to change the height of an item on the screen.

Creating an attractive display is a piece of cake on the Commodore 64 and Vic 20. There's a vast number of ROM graphics symbols and easily embedded cursor and colour control symbols which can be included within parts of any PRINT statement. Take a look at this version of the earlier opening page program:

```
5 POKE 5328Ø,5:POKE 53281,Ø
1Ø PRINT ""TAB(8)
   "□□□□□□□□□□□□□□□□□□□□□□□□□□"
2Ø PRINT TAB(8)"□MARSHALL□
   CAVENDISH LTD□"
3Ø PRINT TAB(1Ø)"□□□□□□□□□□□□□□□□□□□□□□□"
35 PRINT TAB(1Ø)"□COPYRIGHT□
   (C)□1984□"
4Ø PRINT TAB(6)"PRESS SPACE
   BAR TO CONTINUE"
5Ø POKE 198,Ø
6Ø POKE 646,RND(1)*6+2
7Ø IF PEEK(197)=6Ø THEN PRINT
   "":POKE 5328Ø,14:POKE
   53281,6:END
8Ø T=RND(1)*7:PRINT
   ""
   TAB(13+T)""
9Ø PRINT TAB(13+T)"INPUT"
1ØØ PRINT TAB(13+T)
   ""
11Ø PRINT ""
   TAB(11+RND(1)*2)"□P□R□E□
   S□E□N□T□S□":GOTO 6Ø
```

```
5 POKE 36879,11:POKE36865,15Ø:
  POKE 36878,15
1Ø PRINT "MARSHALL CAVENDISH
   LTD"
2Ø PRINT "
   □COPYRIGHT © 1984"
3Ø PRINT TAB(5)"
   PRESS ANY KEY"
4Ø PRINT TAB(6)"TO CONTINUE":
   POKE 198,Ø
5Ø FOR Z=15Ø TO 38 STEP−1:
   POKE 36865,Z:GOSUB 1ØØ
6Ø POKE 36876,255−Z:POKE 36877,
   1ØØ+Z:NEXT:POKE 36877,Ø:POKE
   36876,Ø
7Ø POKE 646,RND(1)*6+2:GOSUB 1ØØ
8Ø IF PEEK(197)<>64 THEN PRINT
   "":POKE 36879,27:END
9Ø GOTO 7Ø
1ØØ T=RND(1)*7:PRINT "
   "TAB(4+T)
   ""
11Ø PRINT TAB(4+T)"INPUT"
12Ø PRINT
   TAB(4+T)""
13Ø PRINT ""TAB
   (2+RND(1)*2)"□P□R□E□
   S□E□N□T□S□":RETURN
```

RUN the program to display a simply constructed but nevertheless eye-catching display which makes good use of the features of the Commodore graphic set.

The first line is the familiar set of POKEs to change border and screen colours. Line 1Ø clears the screen and PRINTs, in yellow, one of the more useful ROM graphics in an application such as this: the thin base line obtained by pressing the @ and C= keys simultaneously. Twenty-four of these are required, adding a thin continuous line at the top of the display MARSHALL CAVENDISH created by the PRINT statement in the line below.

The same technique is used in Line 3Ø to provide a 'topping' for the copyright message in Line 35. Twenty base lines are needed here.

Line 5Ø contains a POKE which clears the keyboard (input) buffer. This is a simple error avoidance technique which stops the program careering onwards if a rogue keypress had been lurking in the buffer when a specific piece of information is awaited.

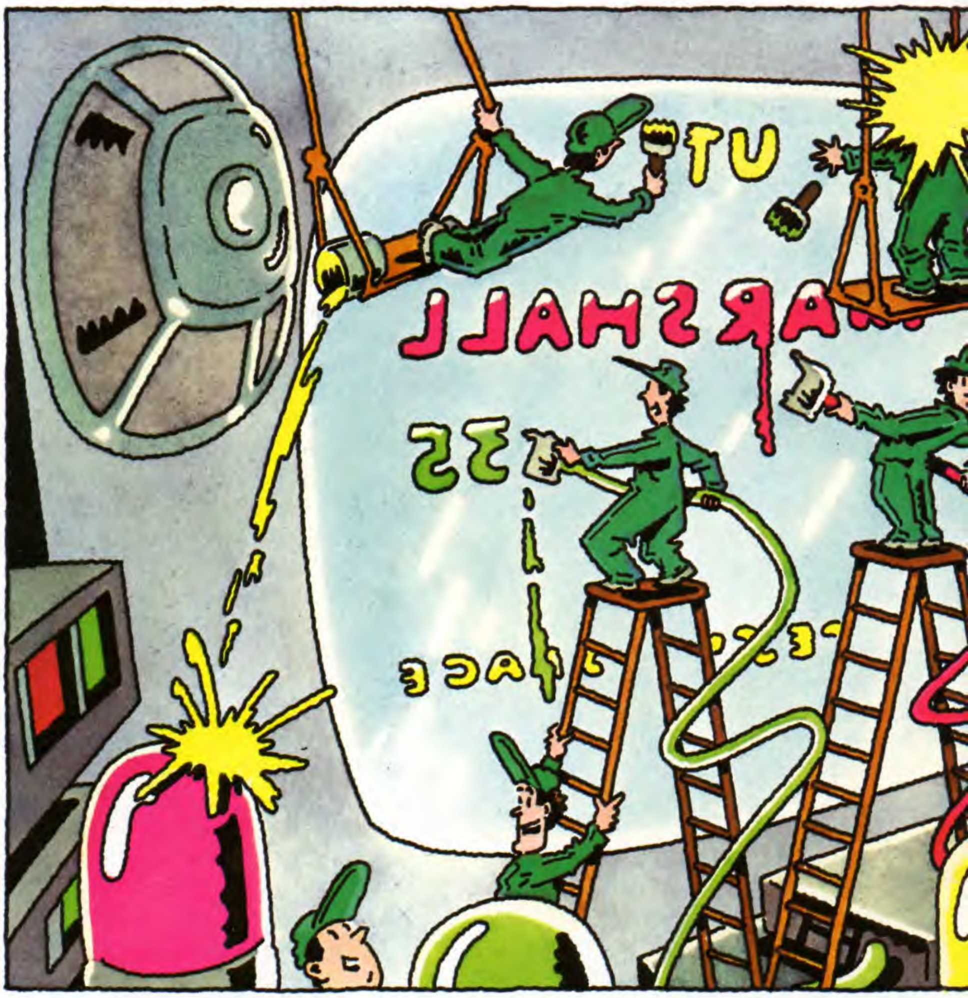
Location 646 used in the POKE in the next line regulates the current character colour. A random value is chosen but within the range which excludes black and white (thus the +2, so you don't get the colour value Ø or 1). Nor does it exceed 8 because too many colour switches in the flashing display routine which follows would be self-defeating.

Line 7Ø waits for the C key to be pressed in response to the main display prompt, clearing and then resetting the screen before terminating the program.

Until the C keypress, the program loops round an interesting variation on a static display. Look at how the TAB value is adjusted by the ever-changing value of T. This gives a shifting effect to the complete INPUT message which is formed by the PRINT statement of Lines 8Ø, 9Ø and 1ØØ.

Finally, in Line 11Ø, is a different version of the variable TAB—an arrangement where the RND element forms part of the argument.

```
10 MODE1
20 VDU19,3,10,0,0,0
30 COLOUR2
40 PRINTTAB(9,2)"MARSHALL CAVENDISH
   LTD"
50 PRINTTAB(15,5)"PRESENTS:"
60 COLOUR3
70 FOR T=0 TO 3
80 PRINTTAB(15,11+T*2)
   "I□N□P□U□T"
90 NEXT
100 COLOUR1
110 PRINTTAB(11,24)"COPYRIGHT (c)1984"
120 PRINTTAB(7,28)"PRESS ANY KEY TO
    CONTINUE"
130 MOVE464,688:DRAW464,432:
    DRAW784,432:DRAW 784,688:
    DRAW 464,688
```



RUN this program and notice the vastly improved title page. The use of colour is accentuated by the use of repetition and by flashing parts of the display. The program starts by selecting a mode that supports four colours, as well as graphics (Line 1Ø). Line 2Ø changes one of the four colours, colour 3, to flashing colour 1Ø, green/magenta. Notice that you could have selected MODE 2 at Line 1Ø which supports all the colours, but that the size and shape of the letters would be rather difficult to read.

The first two lines of the display are positioned and printed (Lines 4Ø and 5Ø) in yellow—selected at Line 3Ø. Line 6Ø then selects the redefined colour, COLOUR3—flashing green/magenta—in which to print the next three lines. These are positioned by the FOR ... NEXT loop between Lines 7Ø and 9Ø. On the first pass round this loop, Line 8Ø PRINTs the word 'INPUT' at position X=15 and Y=11. On subsequent passes, the word is PRINTed two lines down the screen, giving altogether four printings with two lines between each. To PRINT each line in a different colour, you could put in a short routine between Lines 8Ø and 9Ø.

The last two lines are positioned and PRINTed (Lines 11Ø and 12Ø) in red—Line 1ØØ. So far all positions have been selected by PRINT TAB statements. To decide what values they should have, you need to know how many characters are printed across and down the screen in the mode in which you are working. In MODE 1 this is 40 across and 32 down. So if you want to print INPUT at the centre of the screen, you would key PRINT TAB(17,15). These positions are easily reckoned on the Text Planning Sheet included at the back of the User Manual.

The finishing touch to this title page is added by Line 13Ø, which DRAWs a rectangle around the four sides of the word 'INPUT'. You will also find it easy to reckon the position of each of the corners of the rectangle by referring to the Graphics Planning Sheets at the back of the User Manual.

The PRINT TAB statement is useful when you want to place text at a position on the screen, but when you wish to leave spaces to form a pleasant display you need a simpler technique. Enter and RUN the next section of program:

```
14Ø G=GET
15Ø CLS
16Ø PRINTTAB(11,2)"INPUT PART 1 INDEX"
17Ø PRINT''
18Ø FOR T=1 TO 7
19Ø READ A$,B$
2ØØ PRINT'TAB(5)A$TAB(3Ø)B$
21Ø NEXT
22Ø DATA Animation,26-32,Basic
    Programming,2-7,"CHR$, use
    of ",26-27,CLEAR,"1Ø,27",
    "CLOAD,Dragon",14,"CLS,
    explanation of",27,"CODE,
    Spectrum",8
```

The program sets up the title page and stops, but don't be alarmed; there is more. Line 14Ø causes the computer to stop and wait. If you obey the instruction PRINTed by Line 12Ø, the program will continue. The screen is cleared (Line 15Ø) and a title is PRINTed at the top centre (Line 16Ø) of the screen. So that the title stands out and is separated from the rest of the display, Line 17Ø PRINTs three blank lines. To leave a single line space, you would just use PRINT on its own. Notice that you cannot use a double-quotation mark to leave the spaces, or you will get an error.

The rest of the program READs in two columns of data (Line 19Ø) from Line 22Ø. Each time round the FOR ... NEXT loop starting at Line 18Ø, two pieces of data are read, a two line space is left (by PRINT ') and the data is PRINTed. Notice that only one coordinate is used in each TAB statement. In the absence of the second coordinate, the micro takes this to be the X-coordinate, so the first piece of data is PRINTed five units along the X-axis and the second is PRINTed 30 units, also from the left-hand margin.

The DATA statement, too, at Line 22Ø has some important punctuations. Normally, each piece of data is separated by a comma (,). On this line, however, some of the data includes a comma, so to PRINT the data as they are, double quotation marks are used. These are also necessary when you wish to PRINT a BASIC keyword, such as CHR$.

Type in the following program and you'll see how PRINT@ can be used to produce a properly formatted title page.

```
1Ø CLS3:B$=CHR$(128)
2Ø PRINT@68,B$;"marshall";B$;
    "cavendish";B$;"ltd";B$;
3Ø PRINT@138,"PRESENTS:";
4Ø PRINT@358,B$;"copyright";
    B$;B$;B$;"1984";B$;
5Ø PRINT@232,CHR$(19Ø);STRING$
    (11,CHR$(188));CHR$(189);
6Ø PRINT@264,CHR$(234);
    "□I□N□P□U□T□";CHR$(229);
7Ø PRINT@296,CHR$(155);STRING$
    (11,CHR$(147));CHR$(151);
8Ø PRINT@45Ø,"□PRESS ANY KEY TO
    CONTINUE□";
9Ø J=1-J:SCREENØ,J
1ØØ FORK=1TO2ØØ:NEXT
11Ø IF INKEY$<>"" THEN 13Ø
12Ø GOTO 5Ø
13Ø CLS4
14Ø PRINT@7,"input PART 1 INDEX";
15Ø FOR X=3 TO 12
16Ø READ D$,E$
17Ø PRINT@32*X+1,D$;:PRINT@32*X
    +25,LEFT$("□"+E$+"□□□
    □□",7);
18Ø NEXT X
19Ø DATA ANIMATION,26-32,BASIC
    PROGRAMMING,2-7,"BREAK ,dragon",
    7,CASSETTES,25,CASSETTE
    RECORDERS,24,"CHR$ ,use of",
    26-27,CLEAR,"1Ø,27","CLOAD
    ,dragon",14,"CLS ,explanation
    of",27,"CODE ,spectrum",8
2ØØ GOTO 2ØØ
```

The programs starts by changing the screen colour to blue in Line 1Ø—see page 374 for a full explanation of how you can change the screen colour using CLS. The remainder of the line sets B$ equal to a black square.

Lines 2Ø and 4Ø print out the words in reverse characters, using B$ as a space. Remember that the reverse characters on the Dragon and Tandy screen appear as lower case characters on program listings.

Line 3Ø is similar except that the word PRESENTS is less important and so is displayed as normal upper case characters.

In the centre of the screen is the title INPUT surrounded by coloured block graphics. The graphics and title are displayed by Lines 5Ø to 7Ø.

The range of block graphics that are available to you are shown in the user manual. They are made up of black and green areas, but the green parts can be changed to yellow, blue, red, buff, cyan, magenta or orange by adding a multiple of 16 to any of the block graphic character codes.

Parts of the screen display are coloured green. These areas are where the screen background colour shows through. It's very easy to change the green screen colour to orange and produce a flashing screen display. Line 9Ø uses the SCREEN command to swap between the green and orange screen in a very similar fashion to the way you switched between colour sets when using high resolution graphics. By specifying SCREENØ,1 the screen colour can be switched to orange, and by specifying SCREEN Ø,Ø you can switch the screen back to green again. Each time the program executes the loop the screen background colour is changed.

So that you can see the colours alternating a short pause is inserted by Line 1ØØ. The loop is completed by Line 12Ø.

If any key is pressed while the title page is being displayed Line 11Ø causes the program to go on to the next section.

The screen colour is changed to red by Line 13Ø. Line 17Ø and the FOR ... NEXT loop in Lines 15Ø and 18Ø do the work of displaying the data read from Line 19Ø. Each time the loop is executed, a new line is used for the subject and page references. The LEFT$ in the second part of the Line makes sure that the page numbers always appear in the same size panel. The display will look a lot neater that way.

One last nice touch is the loop in Line 2ØØ. This may seem to be absolutely pointless, but its role is quite important. Without Line 2ØØ the program ends causing the message OK to appear against a green screen, ruining your red background.

# A DRAGON/TANDY ASSEMBLER

**Why bother with the tedious translation of assembly language into machine code hex, when that's exactly the sort of mechanical process your computer is good at?**

The assembler for the Dragon and Tandy is a little longer than the one for the Spectrum. But this is because it includes an editor.

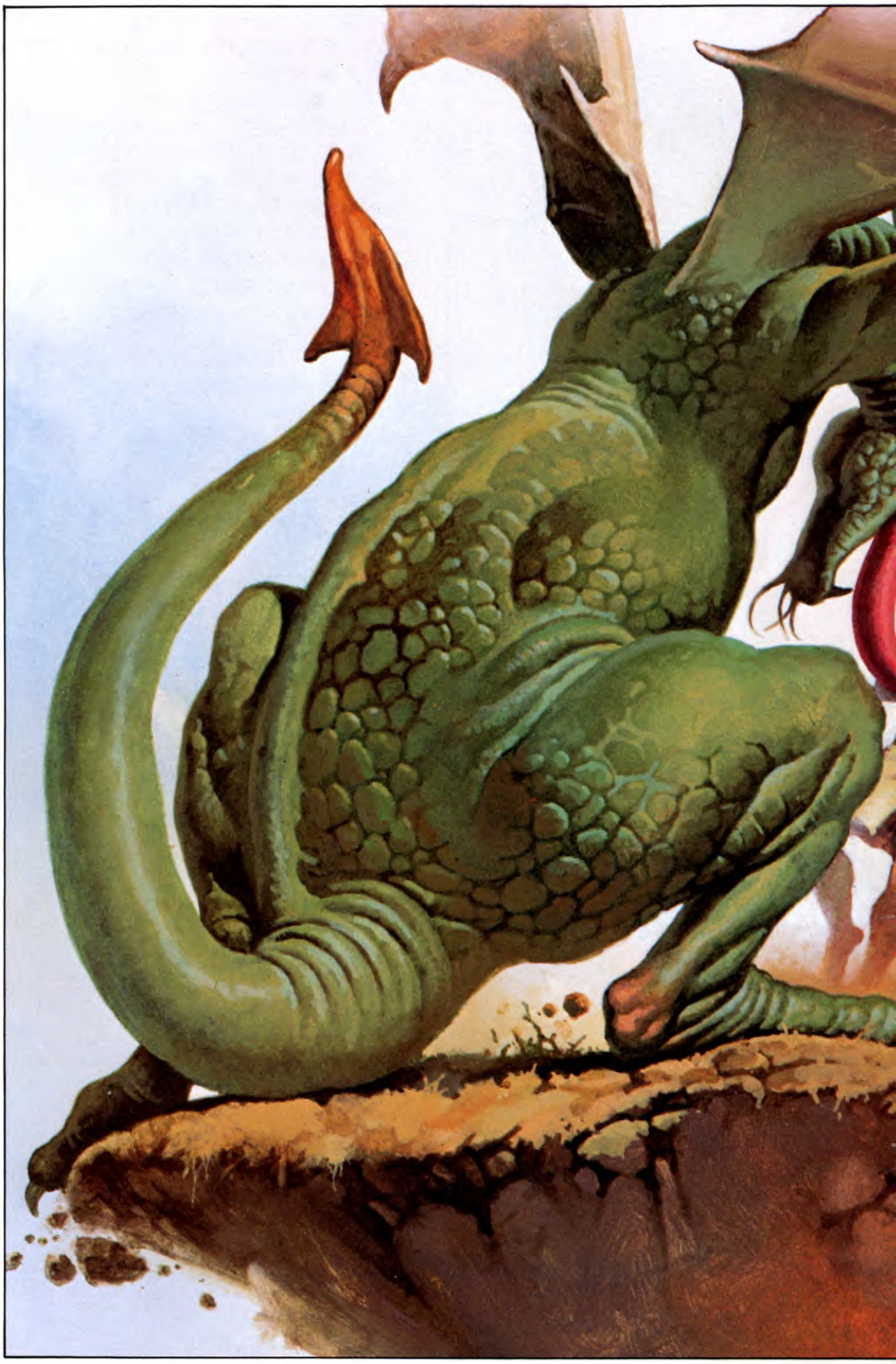
Although there are a good deal fewer instructions for the Dragon's 6809 than for the Spectrum's Z80 chip, the Dragon's assembler has to make three passes instead of the Spectrum's two in case there are any 16-bit instructions.

In spite of the three passes, the Dragon and Tandy assembler is still almost three times faster than the Spectrum's because it uses INSTR to search for the opcodes. But you will still have to wait a bit if you are assembling a long program.

On the Tandy change the POKE in Line 1Ø to POKE 146,1. This line may give an FC error when it is first RUN, but don't worry, just RUN it again and it should be fine.

### THE ASSEMBLER

```
1Ø PMODEØ:PCLEAR1:CLEAR3ØØØ:CLS:
   PRINT@233,"initializing":R$=CHR$(13):
   POKE144,1
2Ø DIMSK$(1),K1(94),K2(94),T$(2ØØ),
   RR(1ØØ),Z$(1ØØ)
3Ø FORCC=1TO94:READK$,K1(CC),K2(CC):
   C=CC/49:SK$(C)=SK$(C)+RIGHT$
   (STR$(CC),2)+K$:NEXT
4Ø DATA ADCA,185,1,ADDA,187,1,ADDD,
   243,2,ASL,12Ø,3,CLR,127,3,CMPA,
   177,1,CMPD,4275,2,CMPY,4284,
   2,BCC,36,4,BCS,37,4,BEQ,39,4
5Ø DATA BHS,36,4,BLO,37,4,BMI,43,4,
   BNE,38,4,BPL,42,4,BRA,32,4,LBRA,
   22,5,BSR,141,4,LBSR,23,5,CMPX,
   188,2,CMPU,4531,2,CMPS,454Ø,2,
   DEC,122,3
6Ø DATA INC,124,3,JSR,189,3,LDA,
   182,1,LDB,246,1,LDD,252,2,LDS,
   435Ø,3,LDU,254,3,LDX,19Ø,3,LDY,
   4286,3,LSL,12Ø,3,LSR,116,3
7Ø DATA PSHS,52,1,PSHU,54,1,PULS,53,
   1,PULU,55,1,ROL,121,3,ROR,118,3,
   RTS,57,,STA,183,3,STB,247,3,STD,
   253,3,STS,4351,3,STU,255,3
8Ø DATA STX,191,3,STY,4287,3,SUBA,
   176,1,SUBD,179,1,ANDA,18Ø,1,ABX,
   58,,ANDCC,76,1,ASR,119,3,RTI,59,,
```



- AUTOMATIC TRANSLATION OF ASSEMBLY LANGUAGE INTO MACHINE CODE
- CALCULATING JUMPS AND BRANCHES
- WORKING OUT POSTBYTES
- COPING WITH LABELS
- **POKE**ING THE HEX INTO MEMORY


```
    SBCA,178,1,NOP,18,,NEG,112,3
90 DATA BITA,181,1,BGE,44,4,BGT,46,
    4,BHI,34,4,BLE,47,4,BLS,35,4,BLT,
    45,4,BRN,33,4,BVC,40,4,BVS,41,4,
    EXG,30,1,TFR,31,1
100 DATA COM,115,3,CWAI,108,1,DAA,
    25,,ORA,186,1,TST,125,3,LEAS,66,3,
    LEAU,67,3,LEAX,64,3,LEAY,65,3,MUL,
    61,,EORA,184,1,ORB,250,1
110 DATA ORCC,74,1,SEX,29,,SWI,63,,
    SWI2,4159,,SWI3,4415,,SYNC,19,,
    EQU,−1,2,FCB,−2,1,FDB,−3,2,RMB,
    −4,,JMP,126,3
120 DIMX$(13),V(14),KK(13),Y$(13)
130 FORC=ØTO12:READX$(C),V(C),KK(C),
    Y$(C):NEXT
140 DATA PCR,253,7,,PC,253,8,D,−−,
    243,1,X,−,242,1,Y,X,159,,U,Y,191,,
    S,U,223,,PC
150 DATA S,255,,,++,241,1,,+,240,1,
    A,A,246,1,B,B,245,1,CC,D,251,1,DP
160 FORJ=ØTO9:READPU$(J),PU(J):NEXT
170 DATA PC,128,U,64,S,64,Y,32,X,16,
    DP,8,D,6,B,4,A,2,CC,1
180 CLS:PRINT@43,"assembler"R$R$
    TAB(8)"G=GET FROM TAPE"R$R$TAB
    (8)"S=SAVE ON TAPE"R$R$TAB(8)
    "A=ASSEMBLE"
190 PRINT@296,"E=EDIT LINE"R$R$
    TAB(8)"D=DELETE LINE"R$R$TAB(8)
    "L=LIST ON SCREEN"
200 A$=INKEY$:IFA$=""THEN200
210 JJ=INSTR("GSEDLA",A$)
220 IFJJ=ØTHENPRINT"<"A$">not
    understood":FORJ=1TO1ØØØ:NEXT:
    GOTO180
230 CLS:ON JJ GOSUB142Ø,145Ø,149Ø,
    171Ø,176Ø,28Ø
240 PRINT@1,"PRESS enter TO
    CONTINUE, ANY□□□□□OTHER
    KEY FOR MAIN MENU"
250 A$=INKEY$:IFA$=""THEN25Ø
260 IFA$<>R$THEN18Ø
270 ON JJ GOSUB142Ø,145Ø,17ØØ,171Ø,
    185Ø,29Ø:GOTO24Ø
280 K=Ø:K9=Ø:PØ=Ø
290 PS=Ø
300 PS=PS+1:IFPS<4THENK=KØ:P=PØ:
    PRINT@1,"START OF PASS"PS:GOTO
    33Ø
310 PØ=P:RETURN
320 IFPS=3 THENPRINT"□postbyte error"
330 GOSUB132Ø
340 GOSUB126Ø:OP$=C$:IFLEFT$(OP$,1)
    ="*"ANDPS=3THENPRINTOP$
350 IFLEFT$(OP$,1)="*"THEN33Ø
360 IFOP$="END"ANDPS=3THENPRINT:
    PRINT"□□□□END LAST ADDR";
    P−1
370 IFOP$="END"THEN3ØØ
380 IFOP$<>"ORG"THEN42Ø
390 GOSUB126Ø:S=Ø:IFLEFT$(C$,1)=
    "*"THENS=P:C$=MID$(C$,2)
400 P=VAL(C$)+S:IFPS=3THENPRINT:
    PRINT"□□□□□ORG";P
410 GOTO33Ø
420 IFP=ØANDPS=3THENPRINT"□you
    forgot org":P=35ØØØ
430 CC=Ø:DF=Ø
440 C=INSTR(SK$(CC),OP$):IFC<>Ø
    THENGOSUB53Ø:GOTO57Ø
450 C=INSTR(SK$(CC),LEFT$(OP$,3)):
    IFC<>ØANDRIGHT$(OP$,1)<"C"AND
    MID$(SK$(CC),C+3,1)<"A"THENGOSUB
    53Ø:GOTO54Ø
460 IFC<>ØANDRIGHT$(OP$,1)="B"
    GOSUB53Ø:GOTO56Ø
470 C=INSTR(SK$(CC),RIGHT$(OP$,3)):
    IFC<>ØANDLEFT$(OP$,1)="L"GOSUB
    53Ø:GOTO55Ø
480 CC=CC+1:IFCC<2THEN44Ø
490 IFPS=3THENPRINTOP$
500 GOSUB135Ø:Q3=Q2:RR(Q2)=P:IFC$=
    ""THEN34Ø
510 IFPS=3THENPRINT"□this line not
    recognized"
520 GOTO33Ø
530 C=VAL(MID$(SK$(CC),C−2,2)):
    RETURN
540 OP=K1(C)−32+16*(RIGHT$(OP$,1)=
    "A"):K2=Ø:GOTO58Ø
550 OP=K1(C)+4Ø96:K2=5:GOTO58Ø
560 OP=K1(C)+64:K2=1:GOTO58Ø
570 OP=K1(C):K2=K2(C)
580 IFPS=3THENBY=P/256:GOSUB124Ø:
    BY=P−32768:GOSUB124Ø:PRINT"□"
    OP$;
590 IFOP>−1THEN74Ø
600 GOSUB126Ø:AD$=C$:IFPS=3 THEN
    PRINT"□"LEFT$(AD$,1Ø);
610 ON −OP□GOTO62Ø,63Ø,66Ø,73Ø
620 GOSUB186Ø:IFNU=ØTHENRR(Q3)=R:
```

```
   GOTO330 ELSE1Ø5Ø
63Ø IFPS=3THENPRINTTAB(2Ø);
64Ø GOSUB69Ø:BY=255ANDR:GOSUB123Ø
65Ø IFB$=AD$THEN33ØELSE64Ø
66Ø IFPS=3THENPRINTTAB(2Ø);
67Ø GOSUB69Ø:BY=INT(R/256):GOSUB
   123Ø:BY=R−256*BY:GOSUB123Ø
68Ø IFB$=AD$THEN33ØELSE67Ø
69Ø NN=INSTR(AD$,“,”):IFNN=ØTHEN
   B$=AD$:GOTO71Ø
7ØØ B$=LEFT$(AD$,NN−1):AD$=MID$
   (AD$,NN+1)
71Ø IFLEFT$(B$,1)=“$”THENR=VAL
   (“&H”+MID$(B$,2))ELSER=VAL(B$)
72Ø RETURN
73Ø GOSUB186Ø:IFNU=ØTHENP=P+R:
   GOTO33Ø ELSE1Ø5Ø
74Ø B=239:IFK2=ØTHEN114Ø
75Ø GOSUB126Ø:AD$=C$:IFPS=3THEN
   PRINT“□”LEFT$(AD$,9);
76Ø IFK2>3THEN1Ø4Ø
77Ø IF (K2<>3OR(LEFT$(OP$,2)=“LD”))
   ANDLEFT$(AD$,1)=“#”THENAD$=
   MID$(AD$,2):DF=1:OP=OP−48:
   GOTO1Ø4Ø
78Ø IFRIGHT$(AD$,1)=“]”THENAD$=
   MID$(AD$,2,LEN(AD$)−2):B=B+16
79Ø IFOP<52OROP>55 THEN87Ø
8ØØ K2=1:R=Ø:AA$=AD$
81Ø NN=INSTR(AA$,“,”):IFNN=ØTHEN
   U$=AA$:GOTO83Ø
82Ø U$=LEFT$(AA$,NN−1):AA$=MID$
   (AA$,NN+1)
83Ø IF U$=RIGHT$(OP$,1)THEN32Ø
84Ø NN=−1:FORJ=ØTO9:IFU$=PU$(J)
   THENNN=J
85Ø NEXT:IFNN<ØTHEN32Ø
86Ø R=R□ORPU(NN):IFU$=AA$THEN
   114ØELSE81Ø
87Ø K2=3:C=INSTR(AD$,“,”):IFC=Ø
   THEN1Ø4Ø
88Ø IFOP<>3ØANDOP<>31THEN95Ø
89Ø NN=INSTR(AD$,“,”):U$=LEFT$
   (AD$,NN−1):GOSUB93Ø:N1=H−1:
   IFH=ØTHEN32Ø
9ØØ U$=MID$(AD$,NN+1):GOSUB93Ø:
   N2=H−1:IFH=ØTHEN32Ø
91Ø IF(8ANDN1)<>(8ANDN2)THEN32Ø
92Ø R=16*N1+N2:K2=1:GOTO114Ø
93Ø FORJ=1TO12:IFY$(J)=U$THENH=J
94Ø NEXT:RETURN
95Ø C2=C+1:FORJ=ØTO12:L=LEN
   (X$(J)):IF MID$(AD$,C2,L)<>X$(J)
   THEN98Ø
96Ø IF (B□ORV(J))AND239<239THEN32Ø
97Ø C2=C2+L:B=B□ANDV(J):IFKK(J)
   THENK2=KK(J)−1
98Ø IFJ=9THENJ2=C2:C2=C2+(C2−1)*
   (K2<6)
99Ø NEXT
1ØØØ IF(15ANDB)=15THENB=B−6
1Ø1Ø IFPS=3ANDJ2<=LEN(AD$)ANDK2
   <>7THENPRINT“ indexing error”:
   GOTO33Ø
1Ø2Ø IF(K2=ØANDC>2)OR(MID$(AD$+
   “,”,J2,1)<>“,”)ANDPS=3THEN
   PRINT“ address error”:GOTO33Ø
1Ø3Ø AD$=LEFT$(AD$,C−1):IFJ2=C
   THENR=Ø:GOTO1Ø6Ø
1Ø4Ø GOSUB186Ø:IFNU=ØTHEN1Ø6Ø
1Ø5Ø IFPS=3THENPRINT“□address not
   understood”
1Ø6Ø IFK2=7THENK2=3:GOTO1Ø8Ø
1Ø7Ø IFK2>3THENR=R−P−2+(OP>
   255)+(B<>239)+(K2>4):
   R=R−((K2=6)AND(R>−129)AND
   (R<128)):K2=K2−3
1Ø8Ø IFB=239ANDK2=3THENK2=2:IFR
   <256ANDDF=ØTHENK2=1:OP=OP−32:
   IF(24ØANDOP)=8ØTHENOP=OP−8Ø
1Ø9Ø IFPS=3AND((OP>31ANDOP<48)OR
   OP=141)AND(R<−128ORR>127)THEN
   PRINT“□branch out of range”;: GOTO33Ø
11ØØ IFB=255THENB=159:K2=2
111Ø IFB<>239THENOP=OP−16:IFK2=3
   THENK2=2:IFABS(R+.5)<128THEN
   K2=1:B=B−1:R=255ANDR
112Ø IF(15ANDB)=8ANDR=ØTHENB=
   B−4:K2=Ø
```

```
1130 IF(31ANDB)=8AND(R<16ORR>239)
   THENB=(B-136)OR(31ANDR):K2=Ø
1140 IFPS=3THENPRINTTAB(2Ø);
1150 IFOP=>ØTHENBY=OP/256:GOSUB
   122Ø:BY=OP:GOSUB123Ø
1160 IFB<>239THENBY=B:GOSUB123Ø
1170 IFK2=ØTHEN33Ø
1180 GOSUB12ØØ:IFK2=2THENBY=R/256:
   GOSUB123Ø
1190 BY=R-256*INT(R/256):GOSUB123Ø:
   GOTO33Ø
1200 IFPS=3THENPRINT"□";
1210 RETURN
1220 IFINT(BY)=ØTHENRETURN
```

```
1230 P=P+1:IFPS=3THENPOKEP-1,255
   ANDBY
1240 BY=255ANDBY:IFPS=3THENPRINT
   RIGHT$("Ø"+HEX$(BY),2);
1250 RETURN
1260 IFK>N THENC$="END":RETURN
1270 K1=K9+1:IFK9>=LEN(T$(K))THEN
   C$="□missing mnemonic":RETURN
1280 K9=K1:IFMID$(T$(K),K1,1)="□"
   THEN127Ø
1290 IFK9>LEN(T$(K))THENC$=MID$
   (T$(K),K1,K9-K1):RETURN
1300 IFMID$(T$(K),K9,1)<>"□"THEN
   K9=K9+1:GOTO129Ø
1310 C$=MID$(T$(K),K1,K9-K1):
   RETURN
1320 IFK9<=LEN(T$(K))ANDPS=3THEN
   PRINTRIGHT$(T$(K),LEN(T$(K))-
   K9+1);
1330 K=K+1:K9=Ø:IFPS=3THENPRINT
1340 RETURN
1350 X$=""
1360 IFC$<"A"ORC$>="["THEN138Ø
1370 X$=X$+LEFT$(C$,1):C$=MID$
   (C$,2):GOTO136Ø
1380 IFC$<>""THENRETURN
1390 FORQ2=1TOVV:IFX$=Z$(Q2)THEN
   141Ø
1400 NEXT:VV=VV+1:Z$(VV)=X$:Q2=VV:
   RR(VV)=23ØØØ
1410 RETURN
1420 CLS:MOTORON:PRINT@161,
   "POSITION TAPE,PRESS ANY KEY AND
   THEN PRESS PLAY ON TAPE"
1430 U$=INKEY$:IFU$=""THEN143Ø
1440 OPEN"I",#-1,"ASM":PRINT
   "□LOADING PROGRAM":INPUT#-1,N:
   FORJ=1TON:INPUT#-1,T$(J):NEXT:
   CLOSE#-1:RETURN
1450 CLS:MOTORON:PRINT@161,
   "POSITION TAPE,PRESS RECORD ON
   □□□TAPE THEN PRESS ANY KEY"
1460 U$=INKEY$:IFU$=""THEN146Ø
1470 OPEN"O",#-1,"ASM":PRINT
   "□SAVING PROGRAM":PRINT#-1,N:
   FORJ=1TON:PRINT#-1,T$(J):NEXT:
   CLOSE#-1:RETURN
1480 PRINT#-1,N:FORJ=1TON:PRINT
   #-1,T$(J):NEXT:CLOSE#-1:
   RETURN
1490 PRINT"□PLEASE INPUT LINE
   NUMBER□□□□□□□(PRESENT
   LINES NUMBERED IN TENS)"
1500 INPUTK:CLS
1510 K2=K/1Ø:IFK2>N□THENK2=N+1:
   N=N+1:T$(K2)="":PRINT@48Ø,""
1520 IFK2<.1THENK2=.1
1530 IFK2=INT(K2)THEN155Ø
1540 K2=INT(K2)+1:FOR K3=N TOK2-1
   STEP-1:T$(K3+1)=T$(K3):NEXT:N=
   N+1:T$(K2)=""
1550 P1=1478:PØ=P1:P2=Ø
1560 PRINT@448-P2,K;TAB(6)T$(K2):
   P9=PØ+LEN(T$(K2))
1565 IFLEN(T$(K2))+PØ>15Ø3THEN
   PØ=PØ-32:P2=P2+32:P1=P1-32:
   GOTO1565
1570 IFP1<PØ THENP1=PØ
1580 IFP1>P9 THENP1=P1-1
1590 P8=PEEK(P1):POKEP1,63ANDP8
1600 P7=Ø:A$=INKEY$:IFA$=""THEN16ØØ
1610 IFA$=R$ THENPOKEP1,P8:RETURN
1620 IFA$=CHR$(9)THENPOKEP1,P8:
   P1=P1+1:GOTO158Ø
1630 IFA$=CHR$(8)THENPOKEP1,P8:
   P1=P1-1:GOTO157Ø
1640 IFA$=CHR$(1Ø)THENA$="":GOTO167Ø
1650 IFA$=CHR$(94)THENA$="□"+
   MID$(T$(K2),P1-PØ+1,1):P7=-1:
   GOTO167Ø
1660 IFA$<"□"THEN16ØØ
1670 IFP1-PØ+1>LEN(T$(K2))THENT$
   (K2)=LEFT$(T$(K2),P1-PØ)+A$:
   GOTO169Ø
1680 T$(K2)=LEFT$(T$(K2),P1-PØ)+A$
   +RIGHT$(T$(K2),LEN(T$(K2))-
   P1+PØ-1)
1690 P1=P1-(LEN(A$)>Ø)+P7:
   GOTO156Ø
1700 PRINT@32,"":K=K+1Ø:GOTO151Ø
1710 IFN=ØTHENCLS:PRINT"□nothing
   to delete":FORC=1TO1ØØØ:NEXT:RETURN
1720 CLS:PRINT"□PLEASE INPUT LINE
   NO□□□□□□□□□□□□
   (PRESENT LINES NUMBERED IN TENS)"
1730 INPUTK:K2=K/1Ø
1740 IFK2>N ORK2<1ORK2<>INT(K2)
   THENPRINT"□THIS LINE DOES NOT
   EXIST":RETURN
1750 K=K2:FORK3=K2 TON:T$(K3)=T$
   (K3+1):NEXT:N=N-1:PRINT@95,
   K*1Ø;"□□";T$(K):RETURN
1760 IFN=ØTHENPRINT"□NOTHING TO
   LIST":FORC=1TO1ØØØ:NEXT:RETURN
1770 PRINT"□PLEASE INPUT FIRST,LAST
   LINE NO(PRESENT LINES NUMBERED
   IN TENS)"
1780 INPUTK,K2:K=INT(K):K2=INT(K2):
   K1=K/1Ø:K2=K2/1Ø
1790 IFK2>N THENK2=N
1800 IFK1<1THENK1=1
1810 IFK2<K1 ANDK2=N THENRETURN
1820 IFK2<K1 THENCLS:PRINT"□BAD
   SET OF LINES":GOTO177Ø
1830 CLS:PRINT@96,;:FORK3=K1 TOK2:
   PRINTK3*1Ø"□"T$(K3):NEXT
1840 RETURN
1850 K=K2-K1:K1=K2+1:K2=K1+K:IF
   N=ØTHEN176ØELSE179Ø
1860 NU=Ø:R=Ø
1870 S=1
1880 IFAD$="" THENRETURN
```

## Microtip

### Debugging long programs

Even the most experienced programmer has trouble keying in a long program like this assembler. No matter how deft your fingers, you are bound to introduce a bug somewhere.

Of course, you'll spot many of these errors when you read over the program. And the computer's own error messages will help you track down others, if you are clever enough to work out what they mean.

But in a long program these error messages are sometimes not enough to help you locate the error in question. A line might be executed many times during a program, but only falter when a variable has been set to an acceptable value by a rogue line elsewhere.

Luckily, the Dragon and the Tandy both have trace programs which will help you diagnose problems with your long programs.

Switch on the trace by keying in TRON. This is a direct statement and does not require a line number. Then RUN your problem program and the trace will fill blank areas of the screen with the numbers of the line of BASIC being executed, as they are executed.

You can compare these against the published program. To switch the trace off, key in TROFF.

```
1890 X$=LEFT$(AD$,1):BD$=MID$
   (AD$,2):IFX$="*"THENR=R+P*S:
   AD$=BD$:GOTO1870
1900 IFX$="+"THENAD$=BD$:GOTO
   1880
1910 IFX$="-"THENAD$=BD$:S=-S:
   GOTO1880
1920 Q=0:IFX$<>"%" THEN1950
1930 IFBD$>="0"ANDBD$<"2"THEN
   Q=Q*2+ASC(BD$)-48:BD$=MID$
   (BD$,2):GOTO1930
1940 R=R+Q*S:AD$=BD$:GOTO1870
1950 IFX$<>"$"ORBD$<"0"ORBD$>
   ="G"THEN1980
1960 Q2=VAL("&H"+LEFT$(BD$,1)):
   BD$=MID$(BD$,2):Q=Q*16+Q2:X$=
   LEFT$(BD$,1)
1970 IF(X$>"/"ANDX$<":")OR(X$>
   "@"ANDX$<"G")THEN1960 ELSER=
   R+Q*S:AD$=BD$:GOTO1870
1980 IFX$<"A"ORX$>"Z"THEN2010
1990 C$=AD$:GOSUB1350:IFC$<>""
   GOSUB1390
2000 R=R+RR(Q2)*S:AD$=C$:GOTO1870
2010 IFX$<"0"ORX$>"9"THENR=0:
   NU=1:RETURN
2020 IFAD$>"/"ANDAD$<":"THENQ=Q*
   10+ASC(AD$)-48:AD$=MID$(AD$,2):
   GOTO2020
2030 R=R+S*Q:GOTO1870
```

## SPEED UP THE ASSEMBLER

It is possible to speed up the assembler by adding the speed poke POKE 65495,0 to the start of Line 180. If you do this you'll also have to change two other lines to slow down the computer again before saving the data to tape. So add POKE 65494,0 to the start of Lines 1420 and 1450.

## HOW IT WORKS

Don't forget to CLEAR enough room for your machine code program before you RUN the assembler (see page 278). When you have keyed in the program RUN it. Once the assembler has initialized, it will display a menu. The various options are spelt out there.

To enter a program press E for edit. The program will then ask you for a line number. With this assembler you have to give a BASIC line number with each instruction. The line numbers should be in tens and there should be no more than one instruction with its associated data and addresses on one line.

The first line should specify an *origin* or starting place for the machine code. This must be above where you have CLEARed to. To specify the origin key in ORG, followed by the start address.

If you don't specify an origin the assembler will take 35,000 as a default value and try to assemble your program in ROM. Of course, it won't work there, as the machine code values can't be POKEd into ROM. But if it defaulted to any other value it might corrupt the program. When it has assembled at the default value, the program tells you 'origin not found'.

Standard 6809 mnemonics are used. Hex numbers should be prefixed with a $, binary numbers should have a % in front. If no prefix is used the assembler will assume that any number it comes across is decimal.

Some assemblers for the Dragon and Tandy will recognize ASCII codes if they are prefixed with ' or !, but ASCII codes cannot be used with this assembler. In consequence, the assembler directive FCC which manipulates ASCII characters cannot be used.

To edit a line before it has been entered, use the cursor controls. The left arrow and right arrow move the cursor along the line. The up arrow can be used to insert something, the down arrow deletes. Otherwise you can simply overwrite the line.

The last line of any program should say END but the assembler will put one in if you forget.

If, instead of pressing ENTER after a line you press another key, the program will take you back to the menu. If you then press L, your mnemonics will be listed for you to check.

If something is wrong, return to the menu and press E again, then specify the number of the line you want to change.

If you want to insert a line, go into the edit mode and give your new line a number which falls between the numbers of the lines you want to put it between. When you list the program again, it will have shoved the subsequent lines along and renumbered them so that they are in tens again. Only insert one line at a time this way though.

To delete a line, go to the main menu and press D, then specify the number of the line you want to delete.

When you are satisfied that the program is okay, return to the main menu and press A. The program will then be assembled. When it has finished it will give you an end address for the program.

The save option—S on the main menu—only saves the assembly language mnemonics, or *source code*. To save the assembler itself, use the normal save routine (see page 23).

To save the machine code program—or *object code*—you have to get out of the program by pressing the BREAK key. Then key in:

```
CSAVEM "NAME",START,END,DEFEXEC
```

Here NAME is the name of the program, which must be in quotes. START is the start address of the program given as the origin. And END is the end address which the assembler gave you when it finished assembling.

The last number you must give it—DEFEXEC—tells the Dragon where to start when you tell it to EXECute the machine code program. In other words it DEFines the EXEC command. Usually this will be the same as the start address.

Now you are ready to assemble any routines and assembly language programs.

## TESTING

To test your assembler try keying in the assembly language right-scrolling program given on page 327. Whether you hand assemble that program or feed it into your assembler, the machine code should read:

```
8E 06 00 E6 82 34 04 C6 1F A6 82 A7 01 5A
26 F9 35 04 E7 84 8C 04 00 2E EA 39
```

An interim index will be published each week. There will be a complete index in the last issue of *INPUT*.

# COMING IN ISSUE 15 . . .

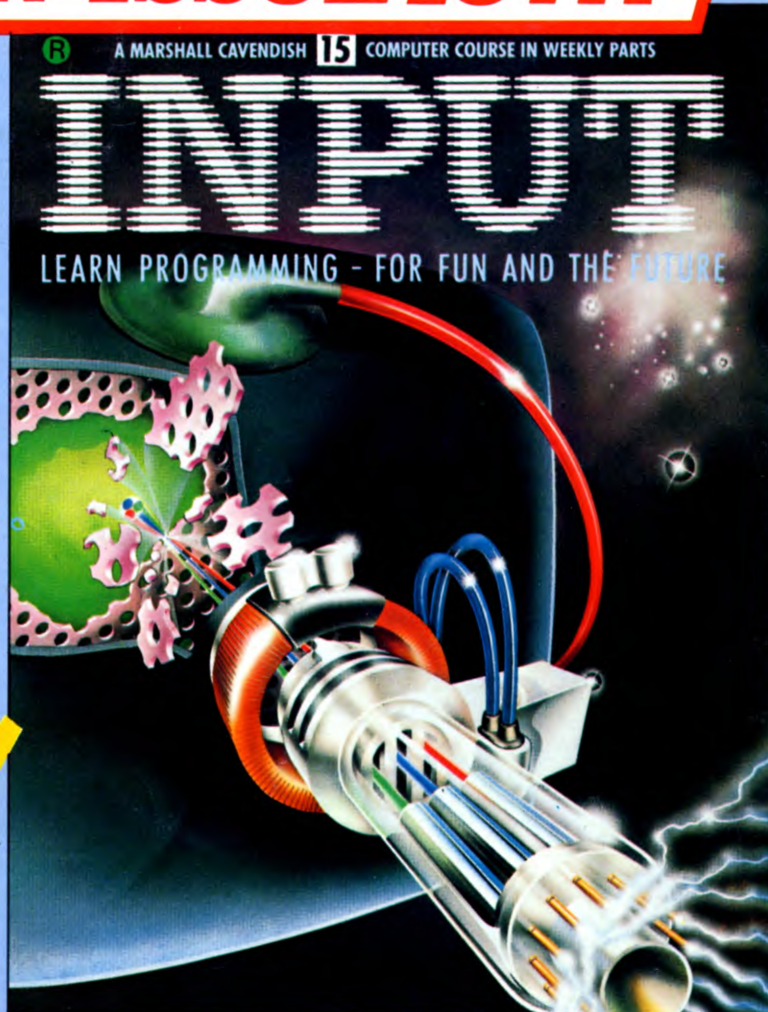
- ***Worried about pirates learning the secrets of your program techniques? Find out how to PROTECT YOUR PROGRAMS.***

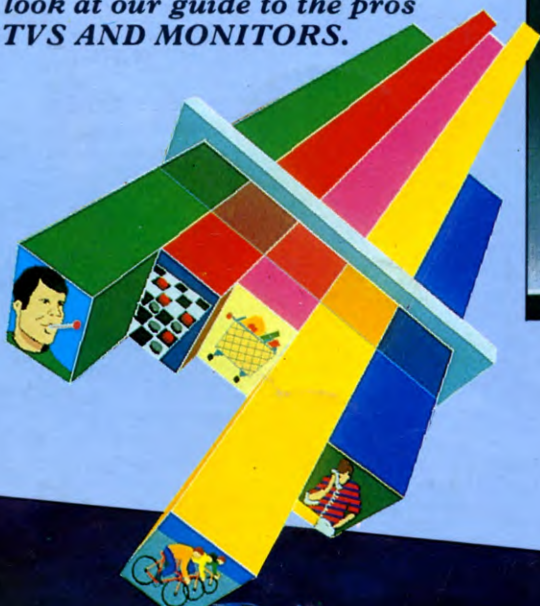
- ***UDGs—versatile and varied. Discover how to use them in large numbers and how to CUSTOMIZE THE CHARACTER SET.***

- ***BAR GRAPHS AND PIE CHARTS are an attractive alternative to linear displays. Learn the techniques of plotting them.***

- ***Enhance the playability of your games programs when you employ routines for JOYSTICK CONTROL FROM BASIC.***

- ***If you've ever had screen problems, take a good look at our guide to the pros and cons of TVS AND MONITORS.***



**ASK YOUR NEWSAGENT FOR INPUT**